

SQL MANUAL

SQL语言使用手册

目录

第 1 章 结构化查询语言 DM_SQL 简介

结构化查询语言 SQL(Structured Query Language)是在 1974 年提出的一种关系数据库语言。由于 SQL 语言接近英语的语句结构，方便简洁、使用灵活、功能强大，倍受用户及计算机工业界的欢迎，被众多计算机公司和数据库厂商所采用，经各公司的不断修改、扩充和完善，SQL 语言最终发展成为关系数据库的标准语言。

SQL 的第一个标准是 1986 年 10 月由美国国家标准化组织(ANSI)公布的 ANSI X3.135-1986 数据库语言 SQL，简称 SQL-86，1987 年国际标准化组织(ISO)也通过了这一标准。以后通过对 SQL-86 的不断修改和完善，于 1989 年第二次公布了 SQL 标准 ISO/IEC 9075-1989(E)，即 SQL-89。1992 年又公布了 SQL 标准 ISO/IEC 9075:1992，即 SQL-92。1999 年公布了 ISO/IEC 9075:1999，即 SQL-3(也称 SQL-99)。之后在 2003 年公布了 ISO/IEC 9075:2003，即 SQL:2003；在 2008 年公布了 ISO/IEC 9075:2008，即 SQL:2008；在 2011 年公布了 ISO/IEC 9075:2011，即 SQL:2011。由于 SQL 标准的内容越来越庞杂，绝大多数情况下，说起 SQL 符合程度，其实是指 SQL-92 中最核心的部分，从 SQL-99 后不再对标准符合程度进行分级，而是改成了核心兼容性和特性兼容性。

SQL 成为国际标准以后，其影响远远超出了数据库领域。例如在 CAD、软件工程、人工智能、分布式等领域，人们不仅把 SQL 作为检索数据的语言规范，而且也把 SQL 作为检索图形、图象、声音、文字等信息类型的语言规范。目前，世界上大型的著名数据库管理系统均支持 SQL 语言，如 Oracle、Sybase、SQL Server、DB2 等。在未来相当长的时间里，SQL 仍将是数据库领域以至信息领域中数据处理的主流语言之一。

由于不同的 DBMS 产品，大都按自己产品的特点对 SQL 语言进行了扩充，很难完全符合 SQL 标准。目前在 DBMS 市场上已将 SQL 的符合率作为衡量产品质量的重要指标，并研制成专门的测试软件，如 NIST。目前，DM 数据库管理系统 SQL-92 入门级符合率达到 100%，过渡级符合率达到 95%，并且部分支持 SQL-99、SQL:2003、SQL:2008 和 SQL:2011 的特性。同时 DM 还兼容 Oracle 11g 和 SQL Server 2008 的部分语言特性。本章主要介绍 DM 系统所支持的 SQL 语言——DM_SQL 语言。

1.1 DM_SQL 语言的特点

DM_SQL 语言符合结构化查询语言 SQL 标准，是标准 SQL 的扩充。它集数据定义、数据查询、数据操纵和数据控制于一体，是一种统一的、综合的关系数据库语言。它功能强大，使用简单方便、容易为用户掌握。DM_SQL 语言具有如下特点：

1. 功能一体化

DM_SQL 的功能一体化表现在以下两个方面：

1) DM_SQL 支持多媒体数据类型，用户在建表时可直接使用。DM 系统在处理常规数据与多媒体数据时达到了四个一体化：一体化定义、一体化存储、一体化检索、一体化处理，最大限度地提高了数据库管理系统处理多媒体的能力和速度；
2) DM_SQL 语言集数据库的定义、查询、更新、控制、维护、恢复、安全等一系列操作于一体，每一项操作都只需一种操作符表示，格式规范，风格一致，简单方便，很容易为用户所掌握。

2． 两种用户接口使用统一语法结构的语言

DM_SQL 语言既是自含式语言，又是嵌入式语言。作为自含式语言，它能独立运行于联机交互方式。作为嵌入式语言，DM_SQL 语句能够嵌入到 C 和 C++语言程序中，将高级语言(也称主语言)灵活的表达能力、强大的计算功能与 DM_SQL 语言的数据处理功能相结合，完成各种复杂的事务处理。而在这两种不同的使用方式中，DM_SQL 语言的语法结构是一致的，从而为用户使用提供了极大的方便性和灵活性。

3． 高度非过程化

DM_SQL 语言是一种非过程化语言。用户只需指出“做什么”，而不需指出“怎么做”，对数据存取路径的选择以及 DM_SQL 语句功能的实现均由系统自动完成，与用户编制的应用程序与具体的机器及关系 DBMS 的实现细节无关，从而方便了用户，提高了应用程序的开发效率，也增强了数据独立性和应用系统的可移植性。

4． 面向集合的操作方式

DM_SQL 语言采用了集合操作方式。不仅查询结果可以是元组的集合，而且一次插入、删除、修改操作的对象也可以是元组的集合，相对于面向记录的数据库语言(一次只能操作一条记录)来说，DM_SQL 语言的使用简化了用户的处理，提高了应用程序的运行效率。

5． 语言简洁，方便易学

DM_SQL 语言功能强大，格式规范，表达简洁，接近英语的语法结构，容易为用户所掌握。

1.2 保留字与标识符

标识符的语法规则兼容标准 GJB 1382A-9X，标识符分为正规标识符和定界标识符两大类。

正规标识符以字母、_、$、#或汉字开头，后面可以跟随字母、数字、_、$、#或者汉字，正规标识符的最大长度是 128 个英文字符或 64 个汉字。正规标识符不能是保留字。

正规标识符的例子：A，test1，_TABLE_B，表 1。

定界标识符的标识符体用双引号括起来时，标识符体可以包含任意字符，特别地，其中使用连续两个双引号转义为一个双引号。

定界标识符的例子："table"，"A"，"!@#$"。

保留字的清单参见附录 1。

1.3 DM_SQL 语言的功能及语句

DM_SQL 语言是一种介于关系代数与关系演算之间的语言，其功能主要包括数据定义、查询、操纵和控制四个方面，通过各种不同的 SQL 语句来实现。按照所实现的功能，DM_SQL 语句分为以下几种：

1． 用户、模式、基表、视图、索引、序列、全文索引、存储过程、触发器等数据库对象的定义和删除语句，数据库、用户、基表、视图、索引、全文索引等数据库对象的修改语句；
2． 查询(含全文检索)、插入、删除、修改语句；
3． 数据库安全语句。包括创建角色语句、删除角色语句，授权语句、回收权限语句，

修改登录口令语句，审计设置语句、取消审计设置语句等。

在嵌入方式中，为了协调 DM_SQL 语言与主语言不同的数据处理方式，DM_SQL 语言引入了游标的概念。因此在嵌入方式下，除了数据查询语句(一次查询一条记录)外，还有几种与游标有关的语句：

1． 游标的定义、打开、关闭、拨动语句；
2． 游标定位方式的数据修改与删除语句。

为了有效维护数据库的完整性和一致性，支持 DBMS 的并发控制机制，DM_SQL 语言提供了事务的回滚(ROLLBACK)与提交(COMMIT)语句。同时 DM 允许选择实施事务级读一致性，它保证同一事务内的可重复读，为此 DM 提供用户多种手动上锁语句，和设置事务隔离级别语句。

1.4 DM_SQL 所支持的数据类型

数据类型是可表示值的集。值的逻辑表示是<字值>。值的物理表示依赖于实现。DM 系统具有 SQL-92 的绝大部分数据类型，以及部分 SQL-99 和 SQL Server 2000 的数据类型。

1.4.1 常规数据类型

1． 字符数据类型

CHAR 类型

语法：CHAR[(长度)]

功能：CHAR 数据类型指定定长字符串。在基表中，定义 CHAR 类型的列时，其最大存储长度由数据库页面大小决定，可以指定一个不超过其最大存储长度的正整数作为字符长度，例如：CHAR(100)。如果未指定长度，缺省为 1。DM 确保存储在该列的所有值都具有这一长度。CHAR 数据类型最大存储长度和页面大小的对应关系请见下表 1.4.1。但是，在表达式计算中，该类型的长度上限不受页面大小限制，为 32767。DM 支持按字节存放字符串。

表 1.4.1

数据库页面大小	实际最大长度
4K	1900
8K	3900
16K	8000
32K	8188

这个限制长度只针对建表的情况，在定义变量的时候，可以不受这个限制长度的限制。另外，实际插入表中的列长度要受到记录长度的约束，每条记录总长度不能大于页面大小的一半。

CHARACTER 类型

语法：CHARACTER[(长度)]

功能：与 CHAR 相同。

VARCHAR 类型/VARCHAR2 类型

语法：VARCHAR[(长度)]

功能：VARCHAR 数据类型指定变长字符串，用法类似 CHAR 数据类型，可以指定一个不超过 8188 的正整数作为字符长度，例如：VARCHAR (100)。如果未指定长度，缺省为 8188。

在基表中，当没有指定 USING LONG ROW 存储选项时，插入 VARCHAR 数据类型的实际最大存储长度由数据库页面大小决定，具体最大长度算法如表 1.4.1;如果指定了 USING LONG ROW 存储选项,则插入 VARCHAR 数据类型的长度不受数据库页面大小限制。VARCHAR 类型在表达式计算中的长度上限不受页面大小限制，为 32767;

CHAR 同 VARCHAR 的区别在于前者长度不足时，系统自动填充空格，而后者只占用实际的字节空间。另外，实际插入表中的列长度要受到记录长度的约束，每条记录总长度不能大于页面大小的一半。

VARCHAR2 类型和 VARCHAR 类型用法相同。

2. 数值数据类型

1. 精确数值数据类型

NUMERIC 类型

```
语法：NUMERIC[(精度 [, 标度])]
```

功能：NUMERIC数据类型用于存储零、正负定点数。其中：精度是一个无符号整数，定义了总的数字数，精度范围是1至38。标度定义了小数点右边的数字位数。一个数的标度不应大于其精度，如果实际标度大于指定标度，那么超出标度的位数将会四舍五入省去。例如：NUMERIC(4,1)定义了小数点前面3位和小数点后面1位，共4位的数字，范围在-999.9到999.9。所有NUMERIC数据类型，如果其值超过精度，DM会返回一个出错信息，如果超过标度，则多余的位会被截断。

如果不指定精度和标度，缺省精度为38，标度无限定。

DECIMAL类型

```
语法：DECIMAL[(精度 [, 标度])]
```

功能：与 NUMERIC 相似。

DEC 类型

```
语法：DEC[(精度[, 标度])]
```

功能：与 DECIMAL 相同。

NUMBER 类型

```
语法：NUMBER[(精度[, 标度])]
```

功能：与 NUMERIC 相同。

INTEGER 类型

```
语法：INTEGER
```

功能：用于存储有符号整数，精度为 10，标度为 0。取值范围为：-2147483648 (-2^{31})～ +2147483647($2^{31}-1$)。

INT 类型

```
语法：INT
```

功能：与 INTEGER 相同。

BIGINT 类型

```
语法：BIGINT
```

功能：用于存储有符号整数，精度为 19，标度为 0。取值范围为：-9223372036854775808(-2^{63})～9223372036854775807($2^{63}-1$)。

TINYINT 类型

```
语法：TINYINT
```

功能：用于存储有符号整数，精度为 3，标度为 0。取值范围为：-128 ～ +127。

BYTE 类型

语法：BYTE

功能：与 TINYINT 相似，精度为 3，标度为 0。

SMALLINT 类型

语法：SMALLINT

功能：用于存储有符号整数，精度为 5，标度为 0。取值范围为：-32768(-2^{15}) ～ +32767($2^{15}-1$)。

BINARY 类型

语法：BINARY[(长度)]

功能：BINARY 数据类型用来存储定长二进制数据。缺省长度为 1 个字节。最大长度由数据库页面大小决定，具体最大存储长度算法与 CHAR 类型的相同，其在表达式计算中的长度上限也与 CHAR 类型相同，为 32767。BINARY 常量以 0x 开始，后面跟着数据的十六进制表示。例如 0x2A3B4058。

VARBINARY 类型

语法：VARBINARY[(长度)]

功能：VARBINARY 数据类型用来存储变长二进制数据，用法类似 BINARY 数据类型，可以指定一个不超过 8188 的正整数作为数据长度。缺省长度为 8188 个字节。VARBINARY 数据类型的实际最大存储长度由数据库页面大小决定，具体最大长度算法与 VARCHAR 类型的相同，其在表达式计算中的长度上限也与 VARCHAR 类型相同，为 32767。

2． 近似数值数据类型

FLOAT 类型

语法：FLOAT[(精度)]

功能：FLOAT 是带二进制精度的浮点数。该类型直接使用标准 C 语言中 DOUBLE。精度值设置无实际意义，精度设置用于保证数据移植的兼容性，实际精度在达梦内部是固定的。精度处于范围（1~126）时忽略精度，超过此范围直接报错。

FLOAT 取值范围$-1.7*10^{308}$ ～ $1.7*10^{308}$。

DOUBLE 类型

语法：DOUBLE[(精度)]

功能：DOUBLE 是带二进制精度的浮点数。DOUBLE 类型的设置是为了移植的兼容性。该类型直接使用标准 C 语言中 DOUBLE。精度与取值范围用法与 FLOAT 完全一样。

REAL 类型

语法：REAL

功能：REAL 是带二进制精度的浮点数，但它不能由用户指定使用的精度，系统指定其二进制精度为 24，十进制精度为 7。取值范围$-3.4*10^{38}$ ～ $3.4*10^{38}$。

DOUBLE PRECISION 类型

语法：DOUBLE PRECISION[(精度)]

功能：该类型指明双精度浮点数。DOUBLE PRECISION 类型的设置是为了移植的兼容性。 该类型直接使用标准 C 语言中 DOUBLE。精度与取值范围用法与 FLOAT 完全一样。

1.4.2 位串数据类型

BIT 类型

语法：BIT

功能：BIT 类型用于存储整数数据 1、0 或 NULL，只有 0 才转换为假，其他非空、非 0 值都会自动转换为真，可以用来支持 ODBC 和 JDBC 的布尔数据类型。DM 的 BIT 类型与 SQL SERVER2000 的 BIT 数据类型相似。

功能与 ODBC 和 JDBC 的 bool 相同。

1.4.3 日期时间数据类型

日期时间数据类型分为一般日期时间数据类型、时间间隔数据类型和时区数据类型三类，用于存储日期、时间和它们之间的间隔信息。

1. 一般日期时间数据类型

DATE 类型

语法：DATE

功能：DATE 类型包括年、月、日信息，定义了'-4712-01-01'和'9999-12-31'之间任何一个有效的格里高利日期。DM 支持儒略历，并考虑了历史上从儒略历转换至格里高利日期时的异常，不计算'1582-10-05'到'1582-10-14'之间的 10 天。

DM 支持 SQL92 标准或 SQL SERVER 的 DATE 字值，例如：DATE '1999-10-01'、'1999/10/01'或'1999.10.01'都是有效的 DATE 值，且彼此等价。年月日中第一个非 0 数值前的 0 亦可省略，例如'0001-01-01'等价于'1-1-1'。

TIME 类型

语法：TIME[(小数秒精度)]

功能：TIME 类型包括时、分、秒信息，定义了一个在'00:00:00.000000'和'23:59:59.999999'之间的有效时间。TIME 类型的小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 0。

DM 支持 SQL92 标准或 SQL SERVER 的 TIME 字值，例如：TIME '09:10:21'，'09:10:21'或'9:10:21'都是有效的 TIME 值，且彼此等价。

TIMESTAMP 类型

语法：TIMESTAMP[(小数秒精度)]

功能：TIMESTAMP 类型包括年、月、日、时、分、秒信息，定义了一个在'-4712-01-01 00:00:00.000000'和'9999-12-31 23:59:59.999999'之间的有效格里高利日期时间。TIMESTAMP 类型的小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。与 DATE 类型相同，DM 不计算'1582-10-05'到'1582-10-14'之间的 10 天。

DM 支持 SQL92 标准或 SQL SERVER 的 TIMESTAMP 字值，例如：TIMESTAMP '2002-12-12 09:10:21'或'2002-12-12 9:10:21'或'2002/12/12 09:10:21'或'2002.12.12 09:10:21' 都是有效的 TIMESTAMP 值，且彼此等价。

语法中，TIMESTAMP 也可以写为 DATETIME。

2. 时间间隔数据类型

DM 支持两类十三种时间间隔类型：两类是年-月间隔类和日-时间隔类，它们通过时间间隔限定符区分，前者结合了日期字段年和月，后者结合了时间字段日、时、分、秒。由时间间隔数据类型所描述的值总是有符号的。

需要说明的是，使用时间间隔数据类型时，如果使用了其引导精度的默认精度，要注意保持精度匹配，否则会出现错误。如果不指定精度，那么将使用默认精度 6。

1) 年-月间隔类

INTERVAL YEAR TO MONTH 类型

语法：INTERVAL YEAR [(引导精度)] TO MONTH

功能：描述一个若干年若干月的间隔，引导精度规定了年的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。月的取值范围在 0 到 11 之间。例如：INTERVAL YEAR(4) TO MONTH，其中 YEAR(4)表示年的精度为 4，表示范围为负 9999 年零 12 月到正 9999 年零 12 月。一个合适的字值例子是：INTERVAL '0015-08' YEAR TO MONTH。

INTERVAL YEAR 类型

语法：INTERVAL YEAR [(引导精度)]

功能：描述一个若干年的间隔，引导精度规定了年的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。例如：INTERVAL YEAR(4)，其中 YEAR(4)表示年的精度为 4，表示范围为负 9999 年到正 9999 年。一个合适的字值例子是：INTERVAL '0015' YEAR。

INTERVAL MONTH 类型

语法：INTERVAL MONTH [(引导精度)]

功能：描述一个若干月的间隔，引导精度规定了月的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。例如：INTERVAL MONTH(4)，其中 MONTH(4)表示月的精度为 4，表示范围为负 9999 月到正 9999 月。一个合适的字值例子是：INTERVAL '0015' MONTH。

2） 日-时间隔类

INTERVAL DAY 类型

语法：INTERVAL DAY [(引导精度)]

功能：描述一个若干日的间隔，引导精度规定了日的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。例如：INTERVAL DAY(3)，其中 DAY (3)表示日的精度为 3，表示范围为负 999 日到正 999 日。一个合适的字值例子是：INTERVAL '150' DAY。

INTERVAL DAY TO HOUR 类型

语法：INTERVAL DAY [(引导精度)] TO HOUR

功能：描述一个若干日若干小时的间隔，引导精度规定了日的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。而时的取值范围在 0 到 23 之间。例如：INTERVAL DAY(1) TO HOUR，其中 DAY(1)表示日的精度为 1，表示范围为负 9 日零 23 小时到正 9 日零 23 小时。一个合适的字值例子是：INTERVAL '9 23' DAY TO HOUR。

INTERVAL DAY TO MINUTE 类型

语法：INTERVAL DAY [(引导精度)] TO MINUTE

功能：描述一个若干日若干小时若干分钟的间隔，引导精度规定了日的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。而小时的取值范围在 0 到 23 之间，分钟的取值范围在 0 到 59 之间。例如：INTERVAL DAY(2) TO MINUTE，其中 DAY(2)表示日的精度为 2，表示范围为负 99 日零 23 小时零 59 分到正 99 日零 23 小时零 59 分。一个合适的字值例子是：INTERVAL '09 23:12' DAY TO MINUTE。

INTERVAL DAY TO SECOND 类型

语法：INTERVAL DAY [(引导精度)] TO SECOND [(小数秒精度)]

功能：描述一个若干日若干小时若干分钟若干秒的间隔，引导精度规定了日的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果不定义小数秒精度默认精度为 6。小时的取值范围在 0 到 23 之间，分钟的取值范围在 0 到 59 之间。例如：INTERVAL DAY(2) TO SECOND(1)，其中 DAY(2)表示日的精度为 2，SECOND(1)表示秒的小数点后面取 1 位，表示范围为负 99 日零 23 小时零 59 分零 59.9 秒到正 99 日零 23 小时零 59 分零 59.9 秒。一个合适的

字值例子是：INTERVAL '09 23:12:01.1' DAY TO SECOND。

INTERVAL HOUR 类型

语法：INTERVAL HOUR [(引导精度)]

功能：描述一个若干小时的间隔，引导精度规定了小时的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。例如：INTERVAL HOUR(3)，其中 HOUR (3)表示时的精度为 3，表示范围为负 999 小时到正 999 小时。一个合适的字值例子是：INTERVAL '150' HOUR。

INTERVAL HOUR TO MINUTE 类型

语法：INTERVAL HOUR [(引导精度)] TO MINUTE

功能：描述一个若干小时若干分钟的间隔，引导精度规定了小时的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。而分钟的取值范围在 0 到 59 之间。例如：INTERVAL HOUR(2) TO MINUTE，其中 HOUR(2)表示小时的精度为 2，表示范围为负 99 小时零 59 分到正 99 小时零 59 分。一个合适的字值例子是：INTERVAL '23:12' HOUR TO MINUTE。

INTERVAL HOUR TO SECOND 类型

语法：INTERVAL HOUR [(引导精度)] TO SECOND [(小数秒精度)]

功能：描述一个若干小时若干分钟若干秒的间隔，引导精度规定了小时的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。分钟的取值范围在 0 到 59 之间。例如：INTERVAL HOUR(2) TO SECOND(1)，其中 HOUR(2)表示小时的精度为 2，SECOND(1)表示秒的小数点后面取 1 位，表示范围为负 99 小时零 59 分零 59.9 秒到正 99 小时零 59 分零 59.9 秒。一个合适的字值例子是：INTERVAL '23:12:01.1' HOUR TO SECOND。

INTERVAL MINUTE 类型

语法：INTERVAL MINUTE [(引导精度)]

功能：描述一个若干分钟的间隔，引导精度规定了分钟的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。例如：INTERVAL MINUTE(3)，其中 MINUTE(3)表示分钟的精度为 3，表示范围为负 999 分钟到正 999 分钟。一个合适的字值例子是：INTERVAL '150' MINUTE。

INTERVAL MINUTE TO SECOND 类型

语法：INTERVAL MINUTE [(引导精度)] TO SECOND [(小数秒精度)]

功能：描述一个若干分钟若干秒的间隔，引导精度规定了分钟的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。例如：INTERVAL MINUTE(2) TO SECOND(1)，其中 MINUTE(2)表示分钟的精度为 2，SECOND(1)表示秒的小数点后面取 1 位，表示范围为负 99 分零 59.9 秒到正 99 分零 59.9 秒。一个合适的字值例子是：INTERVAL '12:01.1' MINUTE TO SECOND。

INTERVAL SECOND 类型

语法：INTERVAL SECOND [(引导精度 [,小数秒精度])]

功能：描述一个若干秒的间隔，引导精度规定了秒整数部分的取值范围。引导精度取值范围为 1～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。例如：INTERVAL SECOND(2,1)，表示范围为负 99.9 秒到正 99.9 秒。一个合适的字值例子是：INTERVAL '51.1' SECOND。

3． 时区数据类型

DM 支持两种时区类型：标准时区类型和本地时区类型。

（一） 标准时区类型：带时区的 TIME 类型和带时区的 TIMESTAMP 类型。

1) TIME WITH TIME ZONE 类型

```
语法：TIME [(小数秒精度)] WITH TIME ZONE
```

功能：描述一个带时区的 TIME 值，其定义是在 TIME 类型的后面加上时区信息。TIME 值部分与 1.4.3 节 1 中的描述一致；时区部分的实质是 INTERVAL HOUR TO MINUTE 类型，取值范围：-12:59 与+14:00 之间。例如：TIME '09:10:21 +8:00'。

DM 支持 SQL92 标准或 SQL SERVER 的 TIME WITH TIME ZONE 字值，例如：TIME '09:10:21 +8:00'，'09:10:21+8:00'或'9:10:21+8:00'都是有效的 TIME WITH TIME ZONE 值，且彼此等价。

例如：

```
CREATE TABLE T2(C1 TIME (2) WITH TIME ZONE);
INSERT INTO T2 VALUES(TIME '09:10:21 +8:00');
```

2) TIMESTAMP WITH TIME ZONE 类型

```
语法：TIMESTAMP [(小数秒精度)] WITH TIME ZONE
```

功能：描述一个带时区的 TIMESTAMP 值，其定义是在 TIMESTAMP 类型的后面加上时区信息。TIMESTAMP 值部分与 1.4.3 节 1 中的描述一致，时区部分的实质是 INTERVAL HOUR TO MINUTE 类型，取值范围：-12:59 与+14:00 之间。例如：'2009-10-11 19:03:05.0000 -02:10'。

DM 支持 SQL92 标准或 SQL SERVER 的 TIMESTAMP 字值，例如：TIMESTAMP '2002-12-12 09:10:21 +8:00'或'2002-12-12 9:10:21 +8:00'或'2002/12/12 09:10:21 +8:00'或'2002.12.12 09:10:21 +8:00' 都是有效的 TIMESTAMP WITH TIME ZONE 值，且彼此等价。

例如：

```
CREATE TABLE T2(C1 TIMESTAMP(2) WITH TIME ZONE);
INSERT INTO T2 VALUES(TIMESTAMP '2002-12-12 09:10:21 +8:00');
```

（二） 本地时区类型：本地时区的 TIMESTAMP 类型。

1) TIMESTAMP WITH LOCAL TIME ZONE 类型

```
语法：TIMESTAMP [(小数秒精度)] WITH LOCAL TIME ZONE
```

功能：描述一个本地时区的 TIMESTAMP 值，能够将标准时区类型 TIMESTAMP WITH TIME ZONE 类型转化为本地时区类型，如果插入的值没有指定时区，则默认为本地时区。

TIMESTAMP 值部分与 1.4.3 节 1 中的描述一致，时区部分的实质是 INTERVAL HOUR TO MINUTE 类型，取值范围：-12:59 与+14:00 之间。例如：'2009-10-11 19:03:05.0000 -02:10'。

DM 支持 SQL92 标准或 SQL SERVER 的 TIMESTAMP 字值，例如：TIMESTAMP '2002-12-12 09:10:21 +8:00'或'2002-12-12 9:10:21 +8:00'或'2002/12/12 09:10:21 +8:00'或'2002.12.12 09:10:21 +8:00' 都是有效的 TIMESTAMP WITH LOCAL TIME ZONE 值，且彼此等价。

例如：

```
CREATE TABLE T2(C1 TIMESTAMP(3) WITH LOCAL TIME ZONE,C2 TIMESTAMP(3) WITH LOCAL TIME ZONE );
INSERT INTO T2 VALUES(TIMESTAMP '2002-12-12 09:10:21 +8:00', TIMESTAMP '2002-12-12 09:10:21');
SELECT * FROM T2;
```

查询结果：

```
2002-12-12 09:10:21.000    2002-12-12 09:10:21.000
```

1.4.4 多媒体数据类型

多媒体数据类型的字值有两种格式：一是字符串，例如：'ABCD'，二是 BINARY，如 0x61626364。

TEXT、LONG、LONGVARCHAR、CLOB 只支持字符串。

BFILE 不适用上面两种格式。BFILE 指明的文件只能只读访问。

不支持为多媒体数据类型的字段指定精度。

1． TEXT 类型

```
语法：TEXT
```

功能：TEXT 为变长字符串类型。其字符串的长度最大为 2G-1。DM 利用它存储长的文本串。

2． LONG、LONGVARCHAR(又名 TEXT)类型

```
语法：LONG/LONGVARCHAR
```

功能：与 TEXT 相同。

3． IMAGE 类型

```
语法：IMAGE
```

功能：IMAGE 用于指明多媒体信息中的图像类型。图像由不定长的象素点阵组成，长度最大为 2G-1 字节。该类型除了存储图像数据之外，还可用于存储任何其它二进制数据。

4． LONGVARBINARY(又名 IMAGE)类型

```
语法：LONGVARBINARY
```

功能：与 IMAGE 相同。

5． BLOB 类型

```
语法：BLOB
```

功能：BLOB 类型用于指明变长的二进制大对象，长度最大为 2G-1 字节。

6． CLOB 类型

```
语法：CLOB
```

功能：CLOB 类型用于指明变长的字母数字字符串，长度最大为 2G-1 字节。

7． BFILE 类型

```
语法：BFILE
```

功能：BFILE 用于指明存储在操作系统中的二进制文件，文件存储在操作系统而非数据库中，仅能进行只读访问。

1.5 DM_SQL 语言支持的表达式

DM 支持多种类型的表达式，包括数值表达式、字符串表达式、时间值表达式、时间间隔值表达式等。本节中引用的数据库实例请参见第 2 章。

1.5.1 数值表达式

1．一元算符 + 和 -

```
语法：+exp 、-exp
```

(exp 代表表达式，下同)

当单独使用时，+ 和 - 代表表达式的正负号。

例

```
select -(-5), +NOWPRICE from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

5　6.1000

注：在 SQL 中由于两短横即"--"，表示"注释开始"，则双负号必须是-(-5)，而不是--5。

2．一元算符 ~

```
语法：~exp
```

按位非算符，要求参与运算的操作数都为整数数据类型。

例

```
select ~10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

-11

3．二元算符 +、-、*、/

```
语法：exp1+exp2 、exp1- exp2 、exp1*exp2 、exp1/exp2
```

(exp1 代表表达式 1，exp2 代表表达式 2，下同)

当在表达式之间使用+、-、*、/ 时，分别表示加、减、乘、除运算。对于结果的精度规定如下：

1) 只有精确数值数据类型的运算

两个相同类型的整数运算的结果类型不变，两个不同类型的整数运算的结果类型转换为范围较大的那个整数类型。

整数与 NUMERIC，DEC 等类型运算时，SMALLINT 类型的精度固定为 5，标度为 0；INTEGER 类型的精度固定为 10，标度为 0；BIGINT 类型的精度固定为 19，标度为 0。

exp1+exp2 结果的精度为二者整数部分长度(即精度－标度)的最大值与二者标度的最大值之和，标度是二者标度的最大值；

exp1-exp2 结果的精度为二者整数部分长度的最大值与二者标度的最大值之和，标度是二者标度的最大值；

exp1*exp2 结果的精度为二者的精度之和，标度也是二者的标度之和；

exp1/exp2 结果为 NUMERIC 类型（缺省的精度为 38，标度未限定)。

例

```
select NOWPRICE+1, NOWPRICE-1 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

7.1 5.1

例

```
select NOWPRICE*10, NOWPRICE/10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

61　0.61

2）　有近似数值数据类型的运算

对于 exp1+exp2 、exp1- exp2 、exp1*exp2 、exp1/exp2 中 exp1 和 exp2 只要有一个为近似数值数据类型，则结果为近似数值数据类型。

例

```
select NOWPRICE+3, NOWPRICE-3, NOWPRICE*3, NOWPRICE/3 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

9.1　　3.1　　18.3　　2.0333333333333333333333333333333334

4．二元算符 &

```
语法：exp1 & exp2
```

按位与算符，要求参与运算的操作数都为整数数据类型。

例

```
select 20 & 10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

0

5．二元算符 |

```
语法：exp1 | exp2
```

按位或算符，要求参与运算的操作数都为整数数据类型。

例

```
select 20 | 10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

30

6．二元算符 ^

```
语法：exp1 ^ exp2
```

按位异或算符，要求参与运算的操作数都为整数数据类型。

例

```
select 20 ^ 10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

30

7．二元算符<<、>>

```
语法：exp1 << exp2
     exp1 >> exp2
```

左移、右移运算符，要求参与运算的操作数只能为整数数据类型、精确数据类型。

例

```
select 1 << 3 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

8

例

```
select  8 >> 3 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

1

当在表达式之间使用<<、>> 时，对于表达式 exp1、exp2 的类型规则如下：

1） 只有整数数据类型的运算

左右操作数都为 smallint、tinyint 时操作结果为 int 类型；

左右操作数都为 int 类型时操作结果为 int 类型；

左右操作数都为 bigint 类型时操作结果为 bigint 类型。

左操作数是 int、smallint、tinyint 类型，右操作数为 int、smallint、tinyint 类型时，操作结果为范围较大的类型。

当左操作数或右操作数有一个是 bigint 类型时，操作结果为 bigint 类型。

2） 有精确数据类型的运算：

当左操作数、右操作数都是精确数据类型时，分别四舍五入转化成 bigint 类型后运算，结果为 bigint 类型；

当整数与精确数据类型运算时，将精确数据类型四舍五入转化成 bigint 类型后运算，结果为 bigint 类型。

3） 有字符串数据类型的运算

字符串指数字字符串，不支持字符串与字符串运算；

当整数与数字字符串数据类型运算时，结果为整数数据类型；

当精确数据类型与字符串数据类型运算时，结果为 bigint 类型。

1.5.2 字符串表达式

连接 ||

```
语法：STR1 || STR2
```

（STR1 代表字符串 1，STR2 代表字符串 2）

连接操作符对两个运算数进行运算，其中每一个都是对属于同一字符集的字符串的求值。它以给定的顺序将字符串连接在一起，并返回一个字符串。其长度等于两个运算数长度之和。如果两个运算数中有一个是 NULL，则 NULL 等价为空串。

例

```
select '武汉' || ADDRESS1 from PERSON.ADDRESS WHERE ADDRESSID=3;
```

结果是：

武汉青山区青翠苑 1 号

1.5.3 时间值表达式

时间值表达式的结果为时间值类型，包括日期(DATE)类型，时间(TIME)类型和时间戳(TIMESTAMP)间隔类型。DM SQL 不是对于任何的日期时间和间隔运算数的组合都可以计算。如果任何一个运算数是 NULL，运算结果也是 NULL。下面列出了有效的可能性和结果的数据类型。

1． 日期+间隔，日期-间隔和间隔+日期，得到日期

日期表达式的计算是根据有效格里高利历日期的规则。如果结果是一个无效的日期，表

达式将出错。参与运算的间隔类型只能是 INTERVAL YEAR、INTERVAL MONTH、INTERVAL YEAR TO MONTH、INTERVAL DAY。

如果间隔运算数是年-月间隔，则没有从运算数的 DAY 字段的进位。

例

```
select PUBLISHTIME + INTERVAL '1' YEAR, PUBLISHTIME - INTERVAL '1' YEAR
from PRODUCTION.PRODUCT
where PRODUCTID=1;
```

结果是：

2006-04-01　2004-04-01

2．时间+间隔，时间-间隔和间隔+时间，得到时间

时间表达式的计算是根据有效格里高利历日期的规则。如果结果是一个无效的时间，表达式将出错。参与运算的间隔类型只能是 INTERVAL DAY、INTERVAL HOUR、INTERVAL MINUTE、INTERVAL SECOND、INTERVAL DAY TO HOUR、INTERVAL DAY TO MINUTE、INTERVAL DAY TO SECOND、INTERVAL HOUR TO MINUTE、INTERVAL HOUR TO SECOND、INTERVAL MINUTE TO SECOND。

当结果的小时值大于等于 24 时，时间表达式是对 24 模的计算。

例

```
SELECT TIME '19:00:00'+INTERVAL '9' HOUR,
TIME '19:00:00'-INTERVAL '9' HOUR;
```

结果是：

04:00:00　10:00:00

3．时间戳记+间隔，时间戳记-间隔和间隔+时间戳记，得到时间戳记

时间戳记表达式的计算是根据有效格里高利历日期的规则。如果结果是一个无效的时间戳记，表达式将出错。参与运算的间隔类型只能是 INTERVAL YEAR、INTERVAL MONTH、INTERVAL YEAR TO MONTH、INTERVAL DAY、INTERVAL HOUR、INTERVAL MINUTE、INTERVAL SECOND、INTERVAL DAY TO HOUR、INTERVAL DAY TO MINUTE、INTERVAL DAY TO SECOND、INTERVAL HOUR TO MINUTE、INTERVAL HOUR TO SECOND、INTERVAL MINUTE TO SECOND。

与时间的计算不同，当结果的小时值大于等于 24 时，结果进位到天。

例

```
SELECT TIMESTAMP'2007-07-15 19:00:00'+INTERVAL'9'HOUR,
TIMESTAMP'2007-07-15 19:00:00'-INTERVAL'9'HOUR;
```

结果是：

2007-07-16 04:00:00 2007-07-15 10:00:00

注：在含有 SECOND 值的运算数之间的一个运算的结果具有等于运算数的小数秒精度的小数秒精度。

4．日期+数值，日期-数值和数值+日期，得到日期

日期与数值的运算，等价于日期与一个 INTERVAL ‘数值’ DAY 的时间间隔的运算。

例

```
SELECT CURDATE();
```

结果是：

2011-09-29 /* 假设该查询操作发生在 2011 年 9 月 29 日 */

例

```
SELECT CURDATE() + 2;
```

结果是：

2011-10-01

例

```
SELECT CURDATE() - 100;
```

结果是：

2011-06-21

5． 时间戳记+数值，时间戳记-数值和数值+时间戳记，得到时间戳记

时间戳记与数值的运算，将数值看作以天为单位，转化为一个 INTERVAL DAY TO SECOND(6)的时间间隔，然后进行运算。

例 时间戳加上 2.358 天。

```
SELECT TIMESTAMP '2003-09-29 08:59:59.123' + 2.358;
```

结果是：

2003-10-01 17:35:30.323000

1.5.4 时间间隔值表达式

1． 日期-日期，得到间隔

由于得到的结果可能会是"年-月-日"间隔，而这是不支持的间隔类型，故要对结果强制使用语法：

(日期表达式-日期表达式)<时间间隔限定符>

结果由<时间间隔限定符>中最不重要的日期字段决定。

例

```
SELECT (PUBLISHTIME-DATE'1990-01-01')YEAR TO MONTH
FROM PRODUCTION.PRODUCT
WHERE PRODUCTID=1;
```

结果是：

INTERVAL '15-3' YEAR(9) TO MONTH

2． 时间一时间，得到间隔

要对结果强制使用语法：

(时间表达式-时间表达式)<时间间隔限定符>

结果由<时间间隔限定符>中最不重要的时间字段决定。

例

```
SELECT (TIME'19:00:00'-TIME'10:00:00') HOUR;
```

结果是：

INTERVAL '9' HOUR(9)

3． 时间戳记 - 时间戳记，得到间隔

要对结果强制使用语法：

(时间戳记表达式-时间戳记表达式)<时间间隔限定符>

结果由<时间间隔限定符>中最不重要的日期时间字段决定。

例

```
SELECT (TIMESTAMP '2007-07-15 19:00:00'- TIMESTAMP '2007-01-15 19:00:00')
HOUR;
```

结果是：

INTERVAL '4344' HOUR(9)

4． 年月间隔 + 年月间隔和年月间隔 - 年月间隔，得到年月间隔

参加运算的两个间隔必须有相同的数据类型，若得出无效的间隔的表达式将出错。结果的子类型包括运算数子类型所有的域，关于结果的引导精度规定如下：

如果二者的子类型相同，则为二者引导精度的最大值；如果二者的子类型不同，则为与结果类型首字段相同的那个运算数的引导精度。

例

```
SELECT INTERVAL'2007-07'YEAR(4) TO MONTH + INTERVAL'7'MONTH,
INTERVAL'2007-07'YEAR(4) TO MONTH - INTERVAL'7'MONTH;
```

结果是：

INTERVAL '2008-2' YEAR(9) TO MONTH INTERVAL '2007-0' YEAR(9) TO MONTH

5． 日时间隔 + 日时间隔和日时间隔 - 日时间隔，得到日时间隔

参加运算的两个间隔必须有相同的数据类型，若得出无效的间隔表达式将出错。结果的子类型包含运算数子类型所有的域，结果的小数秒精度为两运算数的小数秒精度的最大值，关于结果的引导精度规定如下：

如果二者的子类型相同，则为二者引导精度的最大值；如果二者的子类型不同，则为与结果类型首字段相同的那个运算数的引导精度。

例

```
SELECT INTERVAL'7 15'DAY TO HOUR + INTERVAL'10:10'MINUTE TO SECOND,
INTERVAL'7 15'DAY TO HOUR - INTERVAL'10:10'MINUTE TO SECOND;
```

结果是：

INTERVAL '7 15:10:10.000000' DAY(9) TO SECOND(6)

INTERVAL '7 14:49:50.000000' DAY(9) TO SECOND(6)

6． 间隔 * 数字，间隔 / 数字和数字 * 间隔，得到间隔

若得出无效的间隔表达式将出错。结果的子类型、小数秒精度、引导精度和原间隔相同。

例

```
SELECT INTERVAL'10:10'MINUTE TO SECOND * 3,
INTERVAL'10:10'MINUTE TO SECOND / 3;
```

结果是：

INTERVAL '30:30.000000' MINUTE(9) TO SECOND(6)

INTERVAL '3:23.333333' MINUTE(9) TO SECOND(6)

1.5.5 运算符的优先级

当一个复杂的表达式有多个运算符时，运算符优先性决定执行运算的先后次序。运算符有下面这些优先等级(从高到低排列)。在较低等级的运算符之前先对较高等级的运算符进行求值。

```
( )
+(一元正)、-(一元负)、~(一元按位非)
*(乘)、/(除)
+(加)、-(减)
||  (串联)
^(按位异)、&(按位与)、|(按位或)
```

1.6 DM_SQL 语言支持的数据库模式

DM_SQL 语言支持关系数据库的三级模式，外模式对应于视图和部分基表，模式对应于基表，基表是独立存在的表。一个或若干个基表存放于一个存贮文件中，存贮文件中的逻辑结构组成了关系数据库的内模式。DM_SQL 语言本身不提供对内模式的操纵语句。

视图是从基表或其它视图上导出的表，DM 只将视图的定义保存在数据字典中。该定义实际为一查询语句，再为该查询语句取一名字即为视图名。每次调用该视图时，实际上是执行其对应的查询语句，导出的查询结果即为该视图的数据。所以视图并无自己的数据，它是一个虚表，其数据仍存放在导出该视图的基表之中。当基表中的数据改变时，视图中查询的数据也随之改变，因此，视图象一个窗口，用户透过它可看到自己权限内的数据。视图一旦定义也可以为多个用户所共享，对视图作类似于基表的一些操作就像对基表一样方便。

综上所述，SQL 语言对关系数据库三级模式的支持如图 1.6.1 所示。

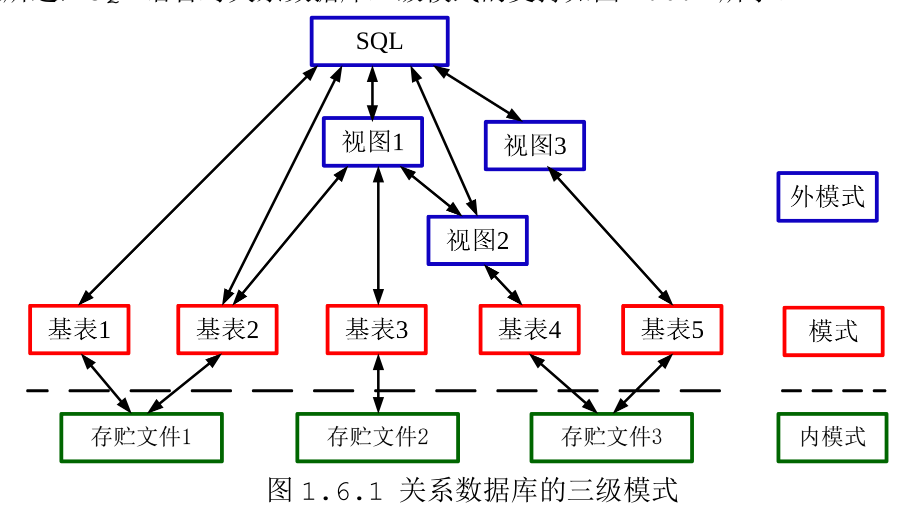


图 1.6.1 关系数据库的三级模式

第 2 章 手册中的示例说明

为方便读者阅读并尽快学会使用 DM 系统，本手册在介绍利用 DM 建立、维护数据库以及对数据库进行的各种操作中，使用了示例库 BOOKSHOP。本章将对该示例库进行说明。

2.1 示例库说明

示例库 BOOKSHOP 模拟武汉代理图书的某销售公司，该公司欲建立在线购物平台来拓展其代理产品的销售渠道，该在线购物平台支持网上产品信息浏览、订购等服务(仅限同城内销售及送货)。该销售公司的雇员、部门信息、业务方案等是所有实例的基础。

安装该示例后，将在数据库中创建 BOOKSHOP 表空间，同时创建 RESOURCES、PERSON、PRODUCTION、PURCHASING、SALES、OTHER 这 6 个模式和相关的表。示例库 BOOKSHOP 的默认数据库名为 DAMENG，默认实例名为 DMSERVER。

示例库 BOOKSHOP 所包含的模式、表如下：

表 2.1.1 BOOKSHOP

模式	包含相关对象	包含的表
RESOURCES	公司基本信息表	EMPLOYEE EMPLOYEE_ADDRESS DEPARTMENT EMPLOYEE_DREPARTMENT
PERSON	各个客户、雇员、供应商的名称和地址	ADDRESS ADDRESS_TYPE PERSON PERSON_TYPE
PRODUCTION	产品销售的信息	PRODUCT PRODUCT_CATEGORY PRODUCT_SUBCATEGORY PRODUCT_INVENTORY LOCATION PRODUCT_REVIEW PRODUCT_VENDOR
PURCHASING	供应商列表(采购订单等)	PURCHASEORDER_HEADER PURCHASEORDER_DETIAL VENDOR VENDOR_ADDRESS VENDOR_PERSON
SALES	与客户销售相关的数据(销售定单等)	CUSTMER CUSTMER_ADDRESS SALESORDER_HEADER SALESORDER_DETAIL SALESPERSON

OTHER	用到的其他表	DEPARTMENT EMPSALARY ACCOUNT ACTIONS READER READERAUDIT DEPTTAB EMPTAB SALGRADE COMPANYHOLIDAYS

EMPLOYEE

该公司的雇员信息表，包含在 RESOURCES 模式中。

表 2.1.2 EMPLOYEE

列	数据类型	是否为空	说明
EMPLOYEEID	INT	非空	主键，自增列
NATIONALNO	VARCHAR(18)	非空	身份证号码
PERSONID	INT	非空	指向 PERSON.PERSONID 的外键
LOGINID	VARCHAR(256)	非空	用户登录 ID
TITLE	VARCHAR(50)	非空	职位
MANAGERID	INT	空	直接上司 ID，指 EMPLOYEE.EMPLOYEEID 的外键
BIRTHDATE	DATE	非空	出生日期
MARITALSTATUS	CHAR(1)	非空	S=未婚 M=已婚
PHOTO	IMAGE	空	照片
HAIRDATE	DATE	非空	入职时间
SALARY	DEC(19,4)	非空	薪资（元）

EMPLOYEE_ADDRESS

将雇员映射到 ADDRESS 表中的地址信息，包含在 RESOURCES 模式中。

表 2.1.3 EMPLOYEE_ADDRESS

列	数据类型	是否为空	说明
EMPLOYEEID	INT	非空	指向 EMPLOYEE.EMPLOYEEID 的外键
ADDRESSID	INT	非空	指向 ADDRESS.ADDRESSID 的外键

EMPLOYEE_DEPARTMENT

将雇员映射到 DEPARTMENT 表中的部门信息，包含在 RESOURCES 模式中。

表 2.1.4 EMPLOYEE_DEPARTMENT

列	数据类型	是否为空	说明
EMPLOYEEID	INT	非空	指向 EMPLOYEE.EMPLOYEEID 的外键
DEPARTMENTID	INT	非空	员工所在部门，指向 DEPARTMENT. DEPARTMENTID 的外键
STARTDATE	DATE	非空	在该部门开始工作的日期
ENDDATE	DATE	空	离开该部门的日期 空=雇员在当前部门

DEPARTMENT

该公司的部门信息，包含在 RESOURCES 模式中。

表 2.1.5 DEPARTMENT

列	数据类型	是否为空	说明
DEPARTENTID	INT	非空	主键， 自增列
NAME	VARCHAR(50)	非空	部门名称

PERSON

该公司所有雇员、供应商、客户的姓名信息表，包含在 PERSON 模式中。

表 2.1.6 PERSON

列	数据类型	是否为空	说明
PERSONID	INT	非空	主键， 自增列，聚集索引
NAME	VARCHAR(50)	非空	姓名
SEX	CHAR(1)	非空	M=男 F=女
EMAIL	VARCHAR(50)	空	电子邮件地址
PHONE	VARCHAR(25)	空	电话

PERSON_TYPE

存储 PERSON 表中的联系人类型，客户中的联系人类型有采购经理、采购代表，供应商的联系人类型有销售经理、销售代表。包含在 PERSON 模式中。

表 2.1.7 PERSON_TYPE

列	数据类型	是否为空	说明
PERSON_TYPEID	INT	非空	主键， 自增列
NAME	VARCHAR(50)	非空	联系人类型说明

ADDRESS

雇员、客户、供应商的地址信息，包含在 PERSON 模式中。

表 2.1.8 ADDRESS

列	数据类型	是否为空	说明
ADDRESSID	INT	非空	主键， 自增列
ADDRESS1	VARCHAR(60)	非空	第一通讯地址
ADDRESS2	VARCHAR(60)	空	第二通讯地址
CITY	VARCHAR(30)	非空	市/县名称
POSTALCODE	VARCHAR(15)	非空	邮政编码

ADDRESS_TYPE

为雇员、客户、供应商定义地址类型的表，包含在 PERSON 模式中。

表 2.1.9 ADDRESS_TYPE

列	数据类型	是否为空	说明
ADDRESS_TYPEID	INT	非空	主键，自增列
NAME	VARCHAR(50)	非空	地址类型的说明如开票地址、家庭地址、送货地址、公司地址等

CUSTOMER

当前客户信息，包含在 SALES 模式中。

表 2.1.10 CUSTOMER

列	数据类型	是否为空	说明
CUSTOMERID	INT	非空	主键， 自增列

PERSONID	INT	非空	指向 PERSON.PERSONID 的外键

CUSTOMER_ADDRESS

将客户映射到某个地址或多个地址，包含在 SALES 模式中。

表 2.1.11 CUSTOMER_ADDRESS

列	数据类型	是否为空	说明
CUSTOMERID	INT	非空	主键，指向 CUSTOMER.CUSTOMERID 的外键
ADDRESSID	INT	非空	主键，指向 ADDRESS.ADDRESSID 的外键
ADDRESS_TYPEID	INT	非空	地址类型，指向 ADDRESS_TYPE.ADDRESS_TYPEID 的外键

SALESPERSON

销售代表的销售统计信息，包含在 SALES 模式中。

表 2.1.12 SALESPERSON

列	数据类型	是否为空	说明
SALESPERSONID	INT	非空	主键， 自增列
EMPLOYEEID	INT	非空	该销售代表对应的雇员号，指向 EMPLOYEE.EMPLOYEEID 的外键
SALESTHISYEAR	DEC(19,4)	非空	今年到目前为止销售总额(万)
SALESLASTYEAR	DEC(19,4)	非空	去年销售总额(万)

VENDOR

所有供应商公司信息，包含在 PURCHASING 模式中。

表 2.1.13 VENDOR

列	数据类型	是否为空	说明
VENDORID	INT	非空	主键， 自增列
ACCOUNTNO	VARCHAR(15)	非空	供应商账户号
NAME	VARCHAR(50)	非空	供应商名称
ACTIVEFLAG	BIT	非空	0=不再使用供应商提供的产品 1=正在使用供应商提供的产品 CHECK 约束
WEBURL	VARCHAR(1024)	空	供应商服务 URL
CREDIT	INT	非空	1=高级 2=很好 3=较好 4=一般 5=较差 CHECK 约束

VENDOR_ADDRESS

所有供应商地址信息，包含在 PURCHASING 模式中。

表 2.1.14 VENDOR_ADDRESS

列	数据类型	是否为空	说明
VENDORID	INT	非空	主键， 指向 VENDOR.VENDORID 的外键
ADDRESSID	INT	非空	主键，指向 ADDRESS.ADDRESSID 外键
ADDRESS_TYPEID	INT	非空	地址类型，指向 ADDRESS_TYPE.ADDRESS_TYPEID 外键

VENDOR_PERSON

供应商联系人信息表。

表 2.1.15 VENDOR_PERSON

列	数据类型	是否为空	说明
VENDORID	INT	非空	主键，指向 VENDOR.VENDORID 外键
PERSONID	INT	非空	主键，指向 PERSON.PERSONID 外键
PERSON_TYPEID	INT	非空	联系人的类型，指向 PERSON_TYPE.PERSON_TYPEID 的外键

PRODUCT

该公司售出的所有产品(图书)，包含在 PRODUCTION 模式中。

表 2.1.16 PRODUCT

列	数据类型	是否为空	说明
PRODUCTID	INT	非空	主键，自增列
NAME	VARCHAR(50)	非空	产品名称
AUTHOR	VARCHAR(25)	非空	作者
PUBLISHER	VARCHAR(50)	非空	出版社
PUBLISHTIME	DATE	非空	出版时间
PRODUCT_SUBCATEGORYID	INT	非空	产品所属子类别，PRODUCT_SUBCATEGORY .PRODUCT_SUBCATEGORYID 的外键
PRODUCTNO	VARCHAR(25)	非空	唯一产品标识号， unique 约束
DESCRIPTION	TEXT	空	产品的说明\简介
PHOTO	IMAGE	空	产品的照片
SATETYSTOCKLEVEL	SMALLINT	非空	最小库存量
ORIGINALPRICE	DEC(19,4)	非空	原价(初始价格)
NOWPRICE	DEC(19,4)	非空	当前销售价格
DISCOUNT	DECIMAL(2,1)	非空	折扣
TYPE	VARCHAR(5)	空	产品开本规格，如 16 开
PAPERTOTAL	INT	空	总页数
WORDTOTAL	INT	空	总字数
SELLSTARTTIME	DATE	非空	开始销售日期
SELLENDTIME	DATE	空	停止销售的日期

PRODUCT_CATEGORY

产品分类表，包含在 PRODUCTION 模式中。

表 2.1.17 PRODUCT_CATEGORY

列	数据类型	是否为空	说明
PRODUCT_CATEGORYID	INT	非空	产品类别 ID，主键，自增列
NAME	VARCHAR(50)	非空	产品类别名称

PRODUCT_SUBCATEGORY

产品子分类表，包含在 PRODUCTION 模式中。

表 2.1.18 PRODUCT_SUBCATEGORY

列	数据类型	是否为空	说明
PRODUCT_SUBCATEGORYID	INT	非空	产品子类别 ID，主键,自增列

PRODUCT_CATEGORYID	INT	非空	产品类别 ID
NAME	VARCHAR(50)	非空	产品类别名称

PRODUCT_INVENTORY

产品的库存信息，包含在 PRODUCTION 模式中。

表 2.1.19 PRODUCT_INVENTORY

列	数据类型	是否为空	说明
PRODUCTID	INT	非空	产品标识，指向 PRODUCT.PRODUCTID 的外键
LOCATIONID	INT	非空	库存位置标识，指向 LOCATION.LOCATIONID 的外键
QUANTITY	INT	非空	库存位置的产品数量

PRODUCT_VENDOR

产品分类表，包含在 PRODUCTION 模式中。

表 2.1.20 PRODUCT_VENDOR

列	数据类型	是否为空	说明
PRODUCTID	INT	非空	主键，指向 PRODUCT.PRODUCTID 外键
VENDORID	INT	非空	主键，指向 VENDOR.VENDORID 外键
STANDARDPRICE	DEC(19,4)	非空	通常价格
LASTPRICE	DEC(19,4)	空	上次采购价格
LASTDATE	DATE	空	上次收到供应商产品的日期
MINQTY	INT	非空	应订购的最小定购数量
MAXQTY	INT	非空	应订购的最大定购数量
ONORDERQTY	INT	空	当前定购的数量

PRODUCT_REVIEW

已售产品的评论表，包含在 PRODCUTION 模式中。

表 2.1.21 PRODUCT_REVIEW

列	数据类型	是否为空	说明
PRODUCT_REVIEWID	INT	非空	主键，自增列
PRODUCTID	INT	非空	指向 PRODUCT.PRODUCTID 外键
NAME	VARCHAR(50)	非空	评论人姓名
REVIEWDATE	DATE	非空	提交评论的日期
EMAIL	VARCHAR(50)	非空	评论人的 EMAIL 地址
RATING	INT	非空	评论人给出的产品等级 范围 1-5，5 为最高级，CHECK 约束
COMMENTS	TEXT	空	评论人的注释

LOCATION

产品的库存地址信息，包含在 PRODUCTION 模式中。

表 2.1.22 LOCATION

列	数据类型	是否为空	说明
LOCATIONID	INT	非空	主键，自增列
NAME	VARCHAR(50)	非空	地点说明
PRODUCT_SUBCATEGORYID	INT	非空	指向 PRODUCT_SUBCATEGORY(PRODUCT_SUBCATEGORYID)的外键

SALESORDER_HEADER

销售订单常规信息表，包含在 SALES 模式中。

表 2.1.23 SALESORDER_HEADER

列	数据类型	是否为空	说明
SALESORDERID	INT	非空	订单 ID，主键，自增列
ORDERDATE	DATE	非空	创建订单的日期
DUEDATE	DATE	非空	客户订单到期的日期
STATUS	TINYINT	非空	订单状态， CHECK 约束 1=待用户确认 2=待发货 3=已发货 4=已完成 5=意外终结
ONLINEORDERFLAG	BIT	非空	0=销售人员下的订单(包括电话订单) 1=在线订单
CUSTOMERID	INT	非空	客户标识号，指向 CUSTOMER.CUSTOMERID 的外键
SALESPERSONID	INT	非空	销售代表 ID，指向 SALESPERSON.SALESPERSONID 的外键
ADDRESSID	INT	非空	客户收货地址，指向 ADDRESS.ADDRESSID 外键
SHIPMETHOD	BIT	非空	发货方法 0=免费送货(仅限中心城区) 1=有偿送货(>=400 元，免费送货；<400，运费 10 元)
SUBTOTAL	DEC(19,4)	非空	销售小计 计算方式:SUM(SALESORDER_DETAIL.LINETOTAL)
FREIGHT	DEC(19,4)	非空	运费(非中城区，subtotal>=400，freight=0；<400，freight=10；中心城区， freight=0)
TOTAL	DEC(19,4)	非空	应付款总计 SUBTOTAL+FREIGHT
COMMENTS	TEXT	空	销售代表备注说明

SALESORDER_DETAIL

与特定销售订单关联的各个产品信息表，包含在 SALES 模式中。

表 2.1.24 SALESORDER_DETAIL

列	数据类型	是否为空	说明
SALESORDERID	INT	非空	主键，指向 SALESORDER_HEADER.SALESORDERID 的外键

SALESORDER_DETAILID	INT	非空	主键，确保数据唯一性的连续编号
CARRIERNO	VARCHAR(25)	非空	发货跟踪号
PRODUCTID	INT	非空	定购产品的 ID，指向 PRODUCT.PRODUCTID 的外键
ORDERQTY	INT	非空	每件产品定购数量
LINETOTAL	DEC(19,4)	非空	产品的销售小计

PURCHASEORDER_HEADER

采购订单常规信息，包含在 PURCHASING 模式中。

表 2.1.25 PURCHASEORDER_HEADER

列	数据类型	是否为空	说明
PURCHASEORDERID	INT	非空	订单 ID，主键，自增列
ORDERDATE	DATE	非空	创建订单的日期
STATUS	TINYINT	非空	订单状态， CHECK 约束 0=等待批准 1=已批准 2=已拒绝 3=已完成
EMPLOYEEID	INT	非空	创建采购订单的雇员 ID，指向 EMPLOYEE.EMPLOYEEID 的外键
VENDORID	INT	非空	所采购产品的供应商 ID，指向 VENDOR.VENDORID 的外键
SHIPMETHOD	VARCHAR(50)	非空	发货方法
SUBTOTAL	DEC(19,4)	非空	产品价格小计
TAX	DEC(19,4)	非空	税额
FREIGHT	DEC(19,4)	非空	运费
TOTAL	DEC(19,4)	非空	应向供应商付款总计(SUBTOTAL+ TAX+ FREIGHT)

PURCHASEORDER_DETAIL

与特定采购订单相关联的每个产品的采购信息，包含在 PURCHASING 模式中。

表 2.1.26 PURCHASEORDER_DETAIL

列	数据类型	是否为空	说明
PURCHASEORDERID	INT	非空	主键，指向 PURCHASEORDER_HEADER .PURCHASEORDERID 的外键
PURCHASEORDER_DETAILID	INT	非空	主键，确保数据唯一性的连续编号
DUEDATE	DATE	非空	希望从供应商收到产品的日期
PRODUCTID	INT	非空	定购产品的 ID，指向 PRODUCT.PRODUCTID 的外键
ORDERQTY	INT	非空	定购数量
PRICE	DEC(19,4)	非空	单件产品的价格
SUBTOTAL	DEC(19,4)	非空	产品价格小计(RPICE*ORDERQTY)
RECEIVEDQTY	DECIMAL(8,2)	非空	实际从供应商收到的数量

REJECTEDQTY	DECIMAL(8,2)	非空	检查时拒收的数量
STOCKEDQTY	DECIMAL(8,2)	非空	纳入库存的数量

2.2 参考脚本

2.2.1 创建示例库

--创建示例库

安装 DM 时，如果选择安装示例库，系统会自动安装一个名为 BOOKSHOP 的示例库。该示例库中，默认数据库名称为 DAMENG，默认实例名为 DMSERVER。

如果安装时没有选择安装示例库，可以通过如下 SQL 语句自行创建：

```
CREATE TABLESPACE BOOKSHOP DATAFILE 'BOOKSHOP.DBF' size 150;
```

2.2.2 创建模式及表

```
--创建模式
CREATE SCHEMA RESOURCES AUTHORIZATION SYSDBA;
/
CREATE SCHEMA PERSON AUTHORIZATION SYSDBA;
/
CREATE SCHEMA SALES AUTHORIZATION SYSDBA;
/
CREATE SCHEMA PRODUCTION AUTHORIZATION SYSDBA;
/
CREATE SCHEMA PURCHASING AUTHORIZATION SYSDBA;
/
CREATE SCHEMA OTHER AUTHORIZATION SYSDBA;
/

--创建表
--CREATE ADDRESS
    CREATE TABLE PERSON.ADDRESS
(ADDRESSID INT IDENTITY(1,1) PRIMARY KEY,
ADDRESS1 VARCHAR(60) NOT NULL,
ADDRESS2 VARCHAR(60),
CITY VARCHAR(30) NOT NULL,
POSTALCODE VARCHAR(15) NOT NULL) STORAGE (on BOOKSHOP);

--CREATE ADDRESS_TYPE
CREATE TABLE PERSON.ADDRESS_TYPE
(ADDRESS_TYPEID INT IDENTITY(1,1) PRIMARY KEY,
NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);
```

```
--CREATE PERSON
CREATE TABLE PERSON.PERSON
(
PERSONID  INT IDENTITY(1,1) CLUSTER PRIMARY KEY,
SEX CHAR(1) NOT NULL,
NAME VARCHAR(50) NOT NULL,
EMAIL VARCHAR(50),
PHONE VARCHAR(25)) STORAGE (ON BOOKSHOP);

--CREATE PERSON_TYPE
CREATE TABLE PERSON.PERSON_TYPE
(PERSON_TYPEID INT IDENTITY(1,1) PRIMARY KEY,
NAME VARCHAR(256) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE DEPARTMENT
CREATE TABLE RESOURCES.DEPARTMENT(DEPARTMENTID INT IDENTITY(1,1) PRIMARY
KEY,NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE EMPLOYEE
CREATE TABLE RESOURCES.EMPLOYEE(
        EMPLOYEEID INT IDENTITY(1,1) PRIMARY KEY ,
        NATIONALNO VARCHAR(18) NOT NULL,
        PERSONID INT NOT NULL REFERENCES PERSON.PERSON(PERSONID),
        LOGINID VARCHAR(256) NOT NULL,
        TITLE VARCHAR(50) NOT NULL,
        MANAGERID INT,
        BIRTHDATE DATE NOT NULL,
        MARITALSTATUS CHAR(1) NOT NULL,
        PHOTO IMAGE,
        HAIRDATE DATE NOT NULL,
        SALARY DEC(19,4) NOT NULL
) STORAGE (ON BOOKSHOP);

--CREATE EMPLOYEE_ADDRESS
CREATE TABLE RESOURCES.EMPLOYEE_ADDRESS
(ADDRESSID INT NOT NULL REFERENCES PERSON.ADDRESS(ADDRESSID),
EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID)) STORAGE
(ON BOOKSHOP);

--CREATE EMPLOYEE_DEPARTMENT
CREATE TABLE RESOURCES.EMPLOYEE_DEPARTMENT
(EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID),
          DEPARTMENTID        INT        NOT        NULL        REFERENCES
```

```
RESOURCES.DEPARTMENT(DEPARTMENTID),
                STARTDATE DATE NOT NULL,
                ENDDATE DATE) STORAGE (ON BOOKSHOP);
    --CREATE CUSTOMER
    CREATE TABLE SALES.CUSTOMER(CUSTOMERID INT IDENTITY(1,1) PRIMARY
KEY ,PERSONID INT NOT NULL REFERENCES PERSON.PERSON(PERSONID)) STORAGE (ON
BOOKSHOP);

    --CREATE CUSTOMER_ADDRESS
    CREATE TABLE SALES.CUSTOMER_ADDRESS
    (CUSTOMERID INT REFERENCES SALES.CUSTOMER(CUSTOMERID),
    ADDRESSID INT REFERENCES PERSON.ADDRESS(ADDRESSID),
    ADDRESS_TYPEID INT NOT NULL REFERENCES PERSON.ADDRESS_TYPE(ADDRESS_TYPEID),
    PRIMARY KEY (CUSTOMERID,ADDRESSID)) STORAGE (ON BOOKSHOP);

    --CREATE PRODUCT_CATEGORY
    CREATE TABLE PRODUCTION.PRODUCT_CATEGORY
    (PRODUCT_CATEGORYID INT IDENTITY(1,1) PRIMARY KEY,
     NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

    --CREATE  PRODUCT_SUBCATEGORY
    CREATE TABLE PRODUCTION.PRODUCT_SUBCATEGORY
    (PRODUCT_SUBCATEGORYID INT IDENTITY(1,1) PRIMARY KEY,
    PRODUCT_CATEGORYID INT NOT NULL ,
    NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

    --CREATE PRODUCT
    CREATE TABLE PRODUCTION.PRODUCT
    (PRODUCTID INT IDENTITY(1,1) PRIMARY KEY,
     NAME VARCHAR(50) NOT NULL,
     AUTHOR VARCHAR(25) NOT NULL,
     PUBLISHER VARCHAR(50) NOT NULL,
     PUBLISHTIME DATE NOT NULL,
     PRODUCT_SUBCATEGORYID INT NOT NULL REFERENCES
PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_SUBCATEGORYID),
     PRODUCTNO VARCHAR(25) NOT NULL,
     SATETYSTOCKLEVEL SMALLINT NOT NULL,
     ORIGINALPRICE DEC(19,4) NOT NULL,
     NOWPRICE DEC(19,4) NOT NULL,
     DISCOUNT DECIMAL(2,1) NOT NULL,
     DESCRIPTION TEXT,
     PHOTO IMAGE,
     TYPE VARCHAR(5),
     PAPERTOTAL INT,
```

```
 WORDTOTAL INT,
 SELLSTARTTIME DATE NOT NULL,
 SELLENDTIME DATE,
 UNIQUE(PRODUCTNO)) STORAGE (ON BOOKSHOP);

--CREATE LOCATION
CREATE TABLE PRODUCTION.LOCATION
(LOCATIONID INT IDENTITY(1,1) PRIMARY KEY,
PRODUCT_SUBCATEGORYID INT NOT NULL REFERENCES
PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_SUBCATEGORYID),
NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PRODUCT_INVENTORY
CREATE TABLE PRODUCTION.PRODUCT_INVENTORY
(PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
LOCATIONID INT NOT NULL REFERENCES PRODUCTION.LOCATION(LOCATIONID),
QUANTITY INT NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PRODUCT_REVIEW
CREATE TABLE PRODUCTION.PRODUCT_REVIEW
(PRODUCT_REVIEWID INT IDENTITY(1,1) PRIMARY KEY,
PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
NAME VARCHAR(50) NOT NULL,
REVIEWDATE DATE NOT NULL,
EMAIL VARCHAR(50) NOT NULL,
RATING INT NOT NULL CHECK(RATING IN(1,2,3,4,5)),
COMMENTS TEXT) STORAGE (ON BOOKSHOP);

--CREATE VENDOR
CREATE TABLE PURCHASING.VENDOR
(VENDORID INT IDENTITY(1,1) PRIMARY KEY,
ACCOUNTNO VARCHAR(15) NOT NULL,
NAME VARCHAR(50) NOT NULL,
ACTIVEFLAG BIT NOT NULL,
WEBURL VARCHAR(1024),
CREDIT INT NOT NULL CHECK(CREDIT IN(1,2,3,4,5))) STORAGE (ON BOOKSHOP);

--CREATE VENDOR_ADDRESS
CREATE TABLE PURCHASING.VENDOR_ADDRESS
(VENDORID INT NOT NULL REFERENCES PURCHASING.VENDOR(VENDORID),
ADDRESSID INT NOT NULL REFERENCES PERSON.ADDRESS(ADDRESSID),
ADDRESS_TYPEID INT NOT NULL REFERENCES PERSON.ADDRESS_TYPE(ADDRESS_TYPEID),
PRIMARY KEY (VENDORID,ADDRESSID)) STORAGE (ON BOOKSHOP);
```

```
--CREATE VENDOR_PERSON
CREATE TABLE PURCHASING.VENDOR_PERSON
(VENDORID INT NOT NULL REFERENCES PURCHASING.VENDOR(VENDORID),
PERSONID INT NOT NULL REFERENCES PERSON.PERSON(PERSONID),
PERSON_TYPEID INT NOT NULL REFERENCES PERSON.PERSON_TYPE(PERSON_TYPEID),
PRIMARY KEY (VENDORID,PERSONID)) STORAGE (ON BOOKSHOP);

--CREATE PRODUCT_VENDOR
CREATE TABLE PRODUCTION.PRODUCT_VENDOR
(PRODUCTID INT REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
 VENDORID INT REFERENCES PURCHASING.VENDOR(VENDORID),
 STANDARDPRICE DEC(19,4) NOT NULL,
 LASTPRICE DEC(19,4),
 LASTDATE DATE,
 MINQTY INT NOT NULL,
 MAXQTY INT NOT NULL,
 ONORDERQTY INT,
 PRIMARY KEY(PRODUCTID,VENDORID)) STORAGE (ON BOOKSHOP);

--CREATE SALESPERSON
CREATE TABLE SALES.SALESPERSON
(SALESPERSONID INT IDENTITY(1,1) PRIMARY KEY,
EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID),
SALESTHISYEAR DEC(19,4) NOT NULL,
SALESLASTYEAR DEC(19,4) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PURCHASEORDER_HEADER
CREATE TABLE PURCHASING.PURCHASEORDER_HEADER
(PURCHASEORDERID INT IDENTITY(1,1) PRIMARY KEY,
ORDERDATE DATE NOT NULL,
STATUS TINYINT NOT NULL CHECK(STATUS IN(0,1,2,3)),
EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID),
VENDORID INT NOT NULL REFERENCES PURCHASING.VENDOR(VENDORID),
SHIPMETHOD VARCHAR(50) NOT NULL,
SUBTOTAL DEC(19,4) NOT NULL,
TAX DEC(19,4) NOT NULL,
FREIGHT DEC(19,4) NOT NULL,
TOTAL DEC(19,4) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PURCHASEORDER_DETAIL
CREATE TABLE PURCHASING.PURCHASEORDER_DETAIL
(PURCHASEORDERID         INT         NOT         NULL         REFERENCES
PURCHASING.PURCHASEORDER_HEADER(PURCHASEORDERID),
PURCHASEORDER_DETAILID INT NOT NULL,
```

```
DUEDATE DATE NOT NULL,
PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
ORDERQTY INT NOT NULL,
PRICE DEC(19,4) NOT NULL,
SUBTOTAL DEC(19,4) NOT NULL,
RECEIVEDQTY INT NOT NULL,
REJECTEDQTY INT NOT NULL,
STOCKEDQTY INT NOT NULL,
PRIMARY KEY(PURCHASEORDERID,PURCHASEORDER_DETAILID)) STORAGE (ON BOOKSHOP);

--CREATE SALESORDER_HEADER
CREATE TABLE SALES.SALESORDER_HEADER
(SALESORDERID INT IDENTITY(1,1) PRIMARY KEY,
ORDERDATE DATE NOT NULL,
DUEDATE DATE NOT NULL,
STATUS TINYINT NOT NULL CHECK(STATUS IN(0,1,2,3,4,5)),
ONLINEORDERFLAG BIT NOT NULL,
CUSTOMERID INT NOT NULL REFERENCES SALES.CUSTOMER(CUSTOMERID),
SALESPERSONID INT NOT NULL REFERENCES SALES.SALESPERSON(SALESPERSONID),
ADDRESSID INT NOT NULL REFERENCES PERSON.ADDRESS(ADDRESSID),
SHIPMETHOD BIT NOT NULL,
SUBTOTAL DEC(19,4) NOT NULL,
FREIGHT DEC(19,4) NOT NULL,
TOTAL DEC(19,4) NOT NULL,
COMMENTS TEXT) STORAGE (ON BOOKSHOP);

--CREATE SALESORDER_DETAIL
CREATE TABLE SALES.SALESORDER_DETAIL
(SALESORDERID INT NOT NULL REFERENCES
SALES.SALESORDER_HEADER(SALESORDERID),
SALESORDER_DETAILID INT NOT NULL,
CARRIERNO VARCHAR(25) NOT NULL,
PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
ORDERQTY INT NOT NULL,
LINETOTAL DEC(19,4) NOT NULL,
PRIMARY KEY(SALESORDERID,SALESORDER_DETAILID)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.DEPARTMENT
CREATE TABLE OTHER.DEPARTMENT
(
HIGH_DEP VARCHAR(50),
DEP_NAME VARCHAR(50)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.EMPSALARY
```

```
CREATE TABLE OTHER.EMPSALARY
(
ENAME CHAR(10),
EMPNO NUMERIC(4),
SAL NUMERIC(4)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.ACCOUNT
CREATE TABLE OTHER.ACCOUNT
(
"ACCOUNT_ID" INTEGER NOT NULL,
"BAL" DEC(10,2),
PRIMARY KEY("ACCOUNT_ID"))STORAGE (ON BOOKSHOP);

--CREATE OTHER.ACTIONS
CREATE TABLE OTHER.ACTIONS
(
"ACCOUNT_ID" INTEGER NOT NULL,
"OPER_TYPE" CHAR(1),
"NEW_VALUE" DEC(10,2),
"STATUS" VARCHAR(50),
PRIMARY KEY("ACCOUNT_ID"))STORAGE (ON BOOKSHOP);

--CREATE OTHER.READER
CREATE TABLE OTHER.READER
(
READER_ID   INT PRIMARY KEY,
NAME        VARCHAR(30),
AGE     SMALLINT,
GENDER      CHAR,
MAJOR       VARCHAR(30)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.READERAUDIT
CREATE TABLE OTHER.READERAUDIT
(
CHANGE_TYPE CHAR NOT NULL,
CHANGED_BY  VARCHAR(8) NOT NULL,
OP_TIMESTAMP    DATE NOT NULL,
OLD_READER_ID   INT,
OLD_NAME    VARCHAR(30),
OLD_AGE     SMALLINT,
OLD_GENDER  CHAR,
OLD_MAJOR   VARCHAR(30),
NEW_READER_ID   INT,
NEW_NAME    VARCHAR(30),
```

```
NEW_AGE     SMALLINT,
NEW_GENDER  CHAR,
NEW_MAJOR   VARCHAR(30)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.DEPTTAB
CREATE TABLE OTHER.DEPTTAB
(
DEPTNO  INT PRIMARY KEY,
DNAME  VARCHAR(15),
LOC  VARCHAR(25)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.EMPTAB
CREATE TABLE OTHER.EMPTAB
(
EMPNO  INT PRIMARY KEY,
ENAME  VARCHAR(15) NOT NULL,
JOB  VARCHAR(10),
SAL  FLOAT,
DEPTNO  INT) STORAGE (on BOOKSHOP);

--CREATE OTHER.SALGRADE
CREATE TABLE OTHER.SALGRADE
(
LOSAL          FLOAT,
HISAL          FLOAT,
JOB_CLASSIFICATION VARCHAR(10)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.COMPANYHOLIDAYS
CREATE TABLE OTHER.COMPANYHOLIDAYS
(
HOLIDAY  DATE) STORAGE (ON BOOKSHOP);
```

2.2.3 插入数据

```
--插入数据
--INSERT ADDRESS
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区 369 号金地太阳城 56-1-202','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区 369 号金地太阳城 57-2-302','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('青山区青翠苑 1 号','','武汉市青山区','430080');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('武昌区武船新村 115 号','','武汉市武昌区','430063');
```

```
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('汉阳大道熊家湾 15 号','','武汉市汉阳区','430050');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区保利花园 50-1-304','','武汉市洪山区','430073');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区保利花园 51-1-702','','武汉市洪山区','430073');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 51-1-702','','武汉市洪山区','430073');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('江汉区发展大道 561 号','','武汉市江汉区','430023');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('江汉区发展大道 555 号','','武汉市江汉区','430023');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('武昌区武船新村 1 号','','武汉市武昌区','430063');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('江汉区发展大道 423 号','','武汉市江汉区','430023');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 55-1-202','','武汉市洪山区','430073');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 10-1-202','','武汉市洪山区','430073');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 11-1-202','','武汉市洪山区','430073');
    INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区光谷软件园 C1_501','','武汉市洪山区','430073');

    --INSERT ADDRESS_TYPE
    INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('发货地址');
    INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('送货地址');
    INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('家庭地址');
    INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('公司地址');

    --INSERT DEPARTMENT
    INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('采购部门');
    INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('销售部门');
    INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('人力资源');
    INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('行政部门');
    INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('广告部');

    --INSERT PERSON
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','李丽','lily@sina.com','02788548562');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','王刚','','02787584562');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','李勇
```

```
',','','02782585462');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','郭艳
','','02787785462');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','孙丽
','','13055173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','黄非
','','13355173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','王菲
','','13255173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','张平
','','13455173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','张红
','','13555173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','刘佳
','','13955173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','王南
','','15955173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','李飞
','','15954173012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','张大海
','','15955673012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','王宇轩
','','15955175012');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','桑泽恩
','','15955173024');
    INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','刘青
','','15955173055');
    INSERT INTO PERSON.PERSON(SEX,NAME,PHONE) VALUES('F','杨凤兰
','02785584662');

    --INSERT PERSON_TYPE
    INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('采购经理');
    INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('采购代表');
    INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('销售经理');
    INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('销售代表');

    --INSERT CUSTOMER
    INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='刘青'));
    INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='桑泽恩'));
    INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='王宇轩'));
    INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
```

```
PERSON.PERSON WHERE NAME='张大海'));
    INSERT  INTO  SALES.CUSTOMER(PERSONID)  VALUES((SELECT  PERSONID  FROM
PERSON.PERSON WHERE NAME='李飞'));
    INSERT  INTO  SALES.CUSTOMER(PERSONID)  VALUES((SELECT  PERSONID  FROM
PERSON.PERSON WHERE NAME='王南'));

    --INSERT  CUSTOMER_ADDRESS
    INSERT INTO SALES.CUSTOMER_ADDRESS(CUSTOMERID,ADDRESSID,ADDRESS_TYPEID)
    SELECT CUSTOMERID,11,2 FROM SALES.CUSTOMER;
    INSERT INTO SALES.CUSTOMER_ADDRESS(CUSTOMERID,ADDRESSID,ADDRESS_TYPEID)
    SELECT CUSTOMERID,12,3 FROM SALES.CUSTOMER;

    --INSERT EMPLOYEE
    INSERT                                                            INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921197908051523',1,'L1','        总        经        理
','','1979-08-05','S','','2002-05-02',40000);

    INSERT                                                            INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921198008051523',2,'L2','销售经理',(SELECT  EMPLOYEEID  FROM
RESOURCES.EMPLOYEE  WHERE  TITLE='总经理'),'1980-08-05','S','','2002-05-02',
26000);

    INSERT                                                            INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921198408051523',3,'L3','采购经理',(SELECT  EMPLOYEEID  FROM
RESOURCES.EMPLOYEE      WHERE      TITLE='      总      经      理
'),'1981-08-05','S','','2002-05-02',23000);

    INSERT                                                            INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921198208051523',4,'L4','销售代表',(SELECT  EMPLOYEEID  FROM
RESOURCES.EMPLOYEE      WHERE      TITLE='      销      售      经      理
'),'1982-08-05','S','','2002-05-02',15000);

    INSERT                                                            INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921198308051523',5,'L5','销售代表',(SELECT  EMPLOYEEID  FROM
```

```
RESOURCES.EMPLOYEE WHERE TITLE='销售经理
'),'1983-08-05','S','','2002-05-02',16000);

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921198408051523',6,'L6','采购代表',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='采购经理
'),'1984-08-05','S','','2005-05-02',12000);

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921197708051523',7,'L7','人力资源部经理',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='总经理
'),'1977-08-05','M','','2002-05-02',25000);

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MAR
ITALSTATUS,PHOTO,HAIRDATE,SALARY)
    VALUES('420921198008071523',8,'L8','系统管理员',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='人力资源部经理
'),'1980-08-07','S','','2004-05-02',20000);
    --INSERT EMPLOYEE_DEPARTMENT
    INSERT INTO
RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'2','2005-02-01',null FROM RESOURCES.EMPLOYEE WHERE
RESOURCES.EMPLOYEE.TITLE='销售代表' OR RESOURCES.EMPLOYEE.TITLE='销售经理';
    INSERT INTO
RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'1','2005-02-01',null FROM RESOURCES.EMPLOYEE WHERE
RESOURCES.EMPLOYEE.TITLE='采购代表' OR RESOURCES.EMPLOYEE.TITLE='采购经理';
    INSERT INTO
RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'3','2005-02-01',null FROM RESOURCES.EMPLOYEE WHERE
RESOURCES.EMPLOYEE.TITLE='系统管理员';
    INSERT INTO
RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'4','2001-02-01',null FROM RESOURCES.EMPLOYEE WHERE
RESOURCES.EMPLOYEE.TITLE='总经理';

    --INSERT EMPLOYEE_ADDRESS
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(1,1);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(2,2);
```

```
INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(3,3);
INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(4,4);
INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(5,5);
INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(6,6);
INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(7,7);
INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(8,8);

--INSERT SALES.SALESPERSON
INSERT INTO SALES.SALESPERSON(EMPLOYEEID,SALESTHISYEAR,SALESLASTYEAR)
SELECT EMPLOYEEID,8,10 FROM RESOURCES.EMPLOYEE WHERE TITLE='销售代表';

--INSERT VENDOR
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','上海画报出版社','1','','2');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','长江文艺出版社','1','','2');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','北京十月文艺出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','人民邮电出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','清华大学出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','中华书局','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','广州出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','上海出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','21 世纪出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','外语教学与研究出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','机械工业出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','文学出版社','1','','1');

--INSERT VENDOR_ADDRESS
INSERT INTO PURCHASING.VENDOR_ADDRESS(VENDORID,ADDRESSID,ADDRESS_TYPEID)
SELECT VENDORID,9,1 FROM PURCHASING.VENDOR;
INSERT INTO PURCHASING.VENDOR_ADDRESS(VENDORID,ADDRESSID,ADDRESS_TYPEID)
SELECT VENDORID,10,4 FROM PURCHASING.VENDOR;

--INSERT VENDOR_PERSON
```

```
    INSERT INTO PURCHASING.VENDOR_PERSON(VENDORID,PERSONID,PERSON_TYPEID)
    SELECT VENDORID,9,4 FROM PURCHASING.VENDOR;

    --INSERT PRODUCT_CATEGORRY
    INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('小说');
    INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('文学');
    INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('计算机');
    INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('英语');
    INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('管理');
    INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('少儿');
    INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('金融');

    --INSERT PRODUCT_SUBCATEGORY
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='小说'),'世界名著');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='小说'),'武侠');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='小说'),'科幻');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='小说'),'四大名著');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='小说'),'军事');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='小说'),'社会');
    INSERT    INTO    PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
VALUES(10,'历史');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='文学'),'文集');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='文学'),'纪实文学');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='文学'),'文学理论');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
```

```
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='文学'),'中国古诗词');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='文学'),'中国现当代诗');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='文学'),'戏剧');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='文学'),'民间文学');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'计算机理论');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'计算机体系结构');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'操作系统');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'程序设计');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'数据库');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'软件工程');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'信息安全');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='计算机'),'多媒体');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='英语'),'英语词汇');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='英语'),'英语语法');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
```

```
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='英语'),'英语听力');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='英语'),'英语口语');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='英语'),'英语阅读');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='英语'),'英语写作');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='管理'),'行政管理');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='管理'),'项目管理');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='管理'),'质量管理与控制');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='管理'),'商业道德');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='管理'),'经营管理');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='管理'),'财务管理');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='少儿'),'幼儿启蒙');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='少儿'),'益智游戏');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='少儿'),'童话');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='少儿'),'卡通');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
```

```
NAME='少儿'),'励志');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY WHERE
NAME='少儿'),'少儿英语');

    --INSERT PRODUCT
    INSERT                                                               INTO
PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCA
TEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,
TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('红楼梦','曹雪芹,高鹗','中华书局','2005-4-1','9787101046120',(SELECT
PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='四大名著
'),'10','19','15.2','8.0','曹雪芹，是中国文学史上最伟大也是最复杂的作家，《红楼梦》也是
中国文学史上最伟大而又最复杂的作品。《红楼梦》写的是封建贵族的青年贾宝玉、林黛王、薛宝钗之间的
恋爱和婚姻悲剧，而且以此为中心，写出了当时具有代表性的贾、王、史、薛四大家族的兴衰，其中又以
贾府为中心，揭露了封建社会后期的种种黑暗和罪恶，及其不可克服的内在矛盾，对腐朽的封建统治阶级
和行将崩溃的封建制度作了有力的批判，使读者预感到它必然要走向覆灭的命运。本书是一部具有高度思
想性和高度艺术性的伟大作品，从本书反映的思想倾向来看，作者具有初步的民主主义思想，他对现实社
会包括宫廷及官场的黑暗，封建贵族阶级及其家庭的腐朽，封建的科举制度、婚姻制度、奴婢制度、等级
制度，以及与此相适应的社会统治思想即孔孟之道和程朱理学、社会道德观念等等，都进行了深刻的批判
并且提出了朦胧的带有初步民主主义性质的理想和主张。这些理想和主张正是当时正在滋长的资本主义经
济萌芽因素的曲折反映。','','16','943','933000','2006-03-20','');

    INSERT                                                               INTO
PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCA
TEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,
TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('水浒传','施耐庵，罗贯中','中华书局','2005-4-1','9787101046137',(SELECT
PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='四大名著
'),'10','19','14.3','7.5','《水浒传》是宋江起义故事在民间长期流传基础上产生出来的，吸收
了民间文学的营养。《水浒传》是我国人民最喜爱的古典长篇白话小说之一。它产生于明代，是在宋、元以
来有关水浒的故事、话本、戏曲的基础上，由作者加工整理、创作而成的。全书以宋江领导的农民起义为
主要题材，艺术地再现了中国古代人民反抗压迫、英勇斗争的悲壮画卷。作品充分暴露了封建统治阶级的
腐朽和残暴，揭露了当时尖锐对立的社会矛盾和"官逼民反"的残酷现实，成功地塑造了鲁智深、李逵、武
松、林冲、阮小七等一批英雄人物。小说故事情节曲折，语言生动，人物性格鲜明，具有高度的艺术成就。
但作品歌颂、美化宋江，鼓吹"忠义"和"替天行道"，表现出严重的思想局限。
','','16','922','912000','2006-03-20','');

    INSERT                                                               INTO
PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCA
TEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,
TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('老人与海','海明威','上海出版社','2006-8-1','9787532740093',(SELECT
PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='世界名著
```

```
'),'10','10','6.1','6.1','海明威(1899 一 1961)，美国著名作家、诺贝尔文学奖获得者。《老人与海》是他最具代表性的作品之一。','','16','98','67000','2006-03-20','');

    INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('射雕英雄传(全四册)','金庸 ','广州出版社','2005-12-1','9787807310822',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='武侠'),'10','32','21.7','6.8','自幼家破人亡的郭靖，随母流落蒙古大漠，这傻头傻脑但有情有义的小伙子倒也逝有福气，他不但习得了江南六怪的绝艺、全真教马钰的内功、洪七公的隆龙十八掌、双手互博之术、九阴真经等盖世武功，还让古灵精怪的小美女黄蓉这辈子跟定了他。这部原名『大漠英雄传』的小说是金庸小说中最广为普罗大众接受、传颂的一部，其中出了许多有名又奇特的人物，东邪西毒南帝北丐中神通，还有武功灵光、脑袋不灵光的老顽童周伯通，他们有特立独行的性格、作为和人生观，让人叹为观止。书中对历史多有着墨，中原武林及蒙古大漠的生活情形随着人物的生长环境变迁而有不同的叙述，异族统治之下的小老百姓心情写来丝丝入扣，本书对情的感觉是很含蓄的，尤其是郭靖与拖雷、华筝无猜的童年之谊，他与江南六怪的师生之谊等等，还有全真七子中长春子丘处机的侠义行为及其与郭杨二人风雪中的一段情谊，也有很豪气的叙述。神算子瑛姑及一灯大师和周伯通的一场孽恋，是最出乎人意料的一段，成人世界的恋情可比小儿女的青涩恋燕还复杂多了。郭靖以扭胜巧的人生经历和「为国为民，侠之大者」的儒侠风范，也是书中最大要旨。距离这本书完成的时间已有四十年了，书中的单纯诚朴的人物性格还深深的留在读者心中，本书故事也多改编成电影、电视剧等，受欢迎程度可见一斑。','','16','','1153000','2006-03-20','');

    INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('鲁迅文集(小说、散文、杂文)全两册','鲁迅','','2006-9-1','9787509000724',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='文集'),'10','39.8','20','5.0','','','16','684','680000','2006-03-20','');

    INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('长征','王树增 ','人民文学出版社','2006-9-1','9787020057986',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='纪实文学'),'10','53','37.7','6.4','','','16','683','670000','2006-03-20','');

    INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
```

```
VALUES('数据结构(C 语言版)(附光盘)','严蔚敏，吴伟民','清华大学出版社','2007-3-1','9787302147510',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='计算机理论'),'10','30','25.5','8.5','《数据结构》(C 语言版)是为"数据结构"课程编写的教材，也可作为学习数据结构及其算法的 C 程序设计的参数教材。本书的前半部分从抽象数据类型的角度讨论各种基本类型的数据结构及其应用','','8','334','493000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
VALUES('工作中无小事','陈满麒','机械工业出版社','2006-1-1','9787111182252',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='行政管理'),'10','16.8','11.4','6.8','本书立足于当今企业中常见的轻视小事，做事浮躁等现象，从人性的弱点这一独特角度，挖掘出员工轻视小事的根本原因，具有深厚的人文关怀，极易引起员工的共鸣。它有助于员工端正心态，摒弃做事贪大的浮躁心理，把小事做好做到位，从而提高整个企业的工作质量。当重视小事成为员工的一种习惯，当责任感成为一种生活态度，他们将会与胜任、优秀、成功同行，责任、忠诚、敬业也将不再是一句空洞的企业宣传口号。本书是一本提升企业竞争力、建设企业文化的指导手册，一本员工素质培训的完美读本，一本所有公务员、公司职员的必读书。','','8','152','70000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
VALUES('突破英文基础词汇','刘毅','外语教学与研究出版社','2003-8-1','9787560035024',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语词汇'),'10','15.9','11.1','7.0','本书所列单词共计 1300 个，加上各词的衍生词、同义词及反义词，则实际收录约 3000 词，均为平时最常用、最容易接触到的单词。详细列出各词的国际音标、词性说明及中文解释，省却查字典的麻烦。每一课分为五个部分，以便于分段记忆。在课前有预备测验，每一部分之后有习题，课后有效果检测，可借助于重复测验来加深对单词的印象，并学习如何活用单词。','','8','350','','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
VALUES('噼里啪啦丛书(全 7 册)','(日)佐佐木洋子','21 世纪出版社','1901-01-01','9787539125992',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='幼儿启蒙'),'10','58','42','6.1','噼里啪啦系列丛书包括：《我要拉巴巴》《我去刷牙》《我要洗澡》《你好》《草莓点心》《车来了》《我喜欢游泳》共 7 册。 这是日本画家佐佐木洋子编绘的，分别描绘孩子在刷牙、洗澡、游玩、吃点心等各种时候所碰到
```

```
的问题，以风趣的方式教会他们人生的最初的知识。书中的图形不仅夸张诱人，而且采用了一些局部折叠的方式，在书页中可以不时翻开一些折叠面，让人看到图画内部的东西，这是很符合低幼儿童的阅读心理的。','','8','','','2006-03-20','');

    --INSERT LOCATION
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='世界名著'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='武侠'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='科幻'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='军事'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='社会'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='文集'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='纪实文学'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='文学理论'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='中国古诗词'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='中国现当代诗'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='戏剧'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='民间文学'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='计算机理论'),'库存 1-货架 2');
```

```
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='计算机体系结构'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='操作系统'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='程序设计'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='数据库'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='软件工程'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='信息安全'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='多媒体'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='英语词汇'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='英语语法'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='英语听力'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='英语口语'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='英语阅读'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='英语写作'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='行政管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
```

```
WHERE NAME='项目管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='质量管理与控制'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='商业道德'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='经营管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='财务管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='幼儿启蒙'),'库存 2-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='益智游戏'),'库存 2-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='童话'),'库存 2-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='卡通'),'库存 2-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='励志'),'库存 2-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE NAME='少儿英语'),'库存 2-货架 2');

    --INSERT PRODUCT_INVENTORY
    INSERT INTO PRODUCTION.PRODUCT_INVENTORY(PRODUCTID,LOCATIONID,QUANTITY)
    SELECT    T1.PRODUCTID,T2.LOCATIONID,100    FROM    PRODUCTION.PRODUCT
T1,PRODUCTION.LOCATION T2
    WHERE T1.PRODUCT_SUBCATEGORYID=T2.PRODUCT_SUBCATEGORYID;

    --INSERT PRODUCT_REVIEW
    INSERT                                                        INTO
PRODUCTION.PRODUCT_REVIEW(PRODUCTID,NAME,REVIEWDATE,EMAIL,RATING,COMMENTS)
    SELECT PRODUCTID,'刘青','2007-05-06','zhangping@sina.com','1','送货快' from
PRODUCTION.PRODUCT;
    INSERT                                                        INTO
```

```
PRODUCTION.PRODUCT_REVIEW(PRODUCTID,NAME,REVIEWDATE,EMAIL,RATING,COMMENTS)
    SELECT PRODUCTID,'桑泽恩','2007-05-06','zhangping@sina.com','1','服务态度好'
from PRODUCTION.PRODUCT;

    --INSERT PRODUCT_VENDOR
    INSERT                                                                 INTO
PRODUCTION.PRODUCT_VENDOR(PRODUCTID,VENDORID,STANDARDPRICE,LASTPRICE,LASTDAT
E,MINQTY,MAXQTY,ONORDERQTY)
    SELECT          PRODUCTID,VENDORID,25,'','','10','100',''              FROM
PRODUCTION.PRODUCT,PURCHASING.VENDOR                                      WHERE
PRODUCTION.PRODUCT.PUBLISHER=PURCHASING.VENDOR.NAME;

    --INSER SALESORDER_HEADER
    INSERT                                                                 INTO
SALES.SALESORDER_HEADER(ORDERDATE,DUEDATE,STATUS,ONLINEORDERFLAG,CUSTOMERID,
SALESPERSONID,ADDRESSID,SHIPMETHOD,SUBTOTAL,FREIGHT,TOTAL,COMMENTS)
    VALUES ('2007-05-06','2007-5-07',2,1,1,2,3,0,36.9,0,36.9,'上午送到');
    INSERT                                                                 INTO
SALES.SALESORDER_HEADER(ORDERDATE,DUEDATE,STATUS,ONLINEORDERFLAG,CUSTOMERID,
SALESPERSONID,ADDRESSID,SHIPMETHOD,SUBTOTAL,FREIGHT,TOTAL,COMMENTS)
    VALUES ('2007-05-07','2007-5-07',1,1,1,1,1,0,36.9,0,36.9,'上午送到');

    -- INSERT SALESORDER_DETAIL
    INSERT                                                                 INTO
SALES.SALESORDER_DETAIL(SALESORDERID,SALESORDER_DETAILID,CARRIERNO,PRODUCTID
,ORDERQTY,LINETOTAL)
    SELECT SALESORDERID,'1','2007052',1,1,15.2 FROM SALES.SALESORDER_HEADER;
    INSERT                                                                 INTO
SALES.SALESORDER_DETAIL(SALESORDERID,SALESORDER_DETAILID,CARRIERNO,PRODUCTID
,ORDERQTY,LINETOTAL)
    SELECT SALESORDERID,'2','2007053',3,1,21.7 FROM SALES.SALESORDER_HEADER;

    UPDATE SALES.SALESPERSON SET SALESLASTYEAR = 20.0000 WHERE SALESPERSONID =
2;

    INSERT                                                                 INTO
PURCHASING.PURCHASEORDER_HEADER(ORDERDATE,STATUS,EMPLOYEEID,VENDORID,SHIPMET
HOD,SUBTOTAL,TAX,FREIGHT,TOTAL)
    VALUES('2006-7-21',1,6,5,'快递',5000.00,600.00,800.00,6400.00);

    -- INSERT DEPARTMENT
    INSERT INTO OTHER.DEPARTMENT VALUES(NULL, '总公司');
    INSERT INTO OTHER.DEPARTMENT VALUES('总公司', '服务部');
    INSERT INTO OTHER.DEPARTMENT VALUES('总公司', '采购部');
```

```
INSERT INTO OTHER.DEPARTMENT VALUES('总公司', '财务部');
INSERT INTO OTHER.DEPARTMENT VALUES('服务部', '网络服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('服务部', '读者服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('服务部', '企业服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('读者服务部', '书籍借阅服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('读者服务部', '书籍阅览服务部');

-- INSERT EMPSALARY
INSERT INTO OTHER.EMPSALARY VALUES ('KING',7839,5000);
INSERT INTO OTHER.EMPSALARY VALUES ('SCOTT',7788,3000);
INSERT INTO OTHER.EMPSALARY VALUES ('FORD',7902,3000);
INSERT INTO OTHER.EMPSALARY VALUES ('JONES',7566,2975);
INSERT INTO OTHER.EMPSALARY VALUES ('BLAKE',7698,2850);
INSERT INTO OTHER.EMPSALARY VALUES ('CLARK',7782,2450);
INSERT INTO OTHER.EMPSALARY VALUES ('ALLEN',7499,1600);
INSERT INTO OTHER.EMPSALARY VALUES ('TURNER',7844,1500);
INSERT INTO OTHER.EMPSALARY VALUES ('MILLER',7934,1300);
INSERT INTO OTHER.EMPSALARY VALUES ('WARD',7521,1250);
INSERT INTO OTHER.EMPSALARY VALUES ('MARTIN',7654,1250);
INSERT INTO OTHER.EMPSALARY VALUES ('ADAMS',7876,1100);
INSERT INTO OTHER.EMPSALARY VALUES ('JAMES',7900,950);
INSERT INTO OTHER.EMPSALARY VALUES ('SMITH',7369,800);

-- INSERT ACCOUNT
INSERT INTO OTHER.ACCOUNT VALUES(1,1000);
INSERT INTO OTHER.ACCOUNT VALUES(2,2000);
INSERT INTO OTHER.ACCOUNT VALUES(3,1500);
INSERT INTO OTHER.ACCOUNT VALUES(4,6500);
INSERT INTO OTHER.ACCOUNT VALUES(5,500);

-- INSERT ACTIONS
INSERT INTO OTHER.ACTIONS VALUES(3,'U',599,NULL);
INSERT INTO OTHER.ACTIONS VALUES(6,'I',20099,NULL);
INSERT INTO OTHER.ACTIONS VALUES(5,'D',NULL,NULL);
INSERT INTO OTHER.ACTIONS VALUES(7,'U',1599,NULL);
INSERT INTO OTHER.ACTIONS VALUES(1,'I',399,NULL);
INSERT INTO OTHER.ACTIONS VALUES(9,'D',NULL,NULL);
INSERT INTO OTHER.ACTIONS VALUES(10,'X',NULL,NULL);

--INSERT READER
INSERT INTO OTHER.READER VALUES(10, 'Bill', 19, 'M', 'Computer');
INSERT INTO OTHER.READER VALUES(11, 'Susan', 18, 'F', 'History');
INSERT INTO OTHER.READER VALUES(12, 'John', 19, 'M', 'Computer');
```

第 3 章 数据定义语句

本章介绍 DM 的数据定义语句，包括数据库修改语句、用户管理语句、模式管理语句、表空间管理语句、表管理语句等等。

需要注意的是，在数据定义语句中有时需要指定一些文件的路径，无论用户指定的是绝对路径还是相对路径，DM 在处理时最终都会将其统一处理为绝对路径，DM 规定这个绝对路径的长度不能超过 256 字节。

3.1 数据库修改语句

一个数据库创建成功后，可以增加和重命名日志文件，修改日志文件大小；可以修改数据库的状态和模式；还可以进行归档配置。

语法格式

```
ALTER DATABASE <修改数据库语句>;
<修改数据库语句>::=
    RESIZE LOGFILE <文件路径> TO <文件大小>|
    ADD LOGFILE <文件说明项>{,<文件说明项>}|
    ADD NODE LOGFILE <文件说明项>,<文件说明项>{,<文件说明项>}|
    RENAME LOGFILE <文件路径>{,<文件路径>} TO <文件路径>{,<文件路径>}|
    MOUNT |
    SUSPEND |
    OPEN [FORCE] |
    NORMAL |
    PRIMARY|
    STANDBY |
    ARCHIVELOG |
    NOARCHIVELOG |
    <ADD|MODIFY|DELETE> ARCHIVELOG <归档配置语句> |
    ARCHIVELOG CURRENT
<文件说明项> ::= <文件路径>SIZE <文件大小>
<归档配置语句>::= 'DEST = <归档目标>,TYPE = <归档类型>'
<归档类型>::=
    LOCAL [<文件和路径设置>] |
    REALTIME|
    ASYNC ,TIMER_NAME = <定时器名称> |
    REMOTE ,INCOMING_PATH = <远程归档路径>[<文件和路径设置>]|
    TIMELY
<文件和路径设置>::=[,FILE_SIZE = <文件大小>][,SPACE_LIMIT = <空间大小限制>]
```

参数

1. <文件路径> 指明被操作的数据文件在操作系统下的绝对路径：'路径+数据文件名'。例如：'C:\DMDBMS\data\dmlog_0.log'；

2. <文件大小> 整数值，单位为 M；

3. <归档目标> 指归档日志所在位置，若本地归档，则本地归档目录；若远程归档，则为远程服务实例名；删除操作，只需指定归档目标；

4. <归档类型> 指归档操作类型，包括 REALTIME/ASYNC/LOCAL/REMOTE/TIMELY，分别表示远程实时归档/远程异步归档 /本地归档/远程归档/主备即时归档；

5. <空间大小限制> 整数值，范围（1024~4294967294），若设为 0，表示不限制，仅本地归档有效；

6. <定时器名称> 异步归档中指定的定时器名称，仅异步归档有效。

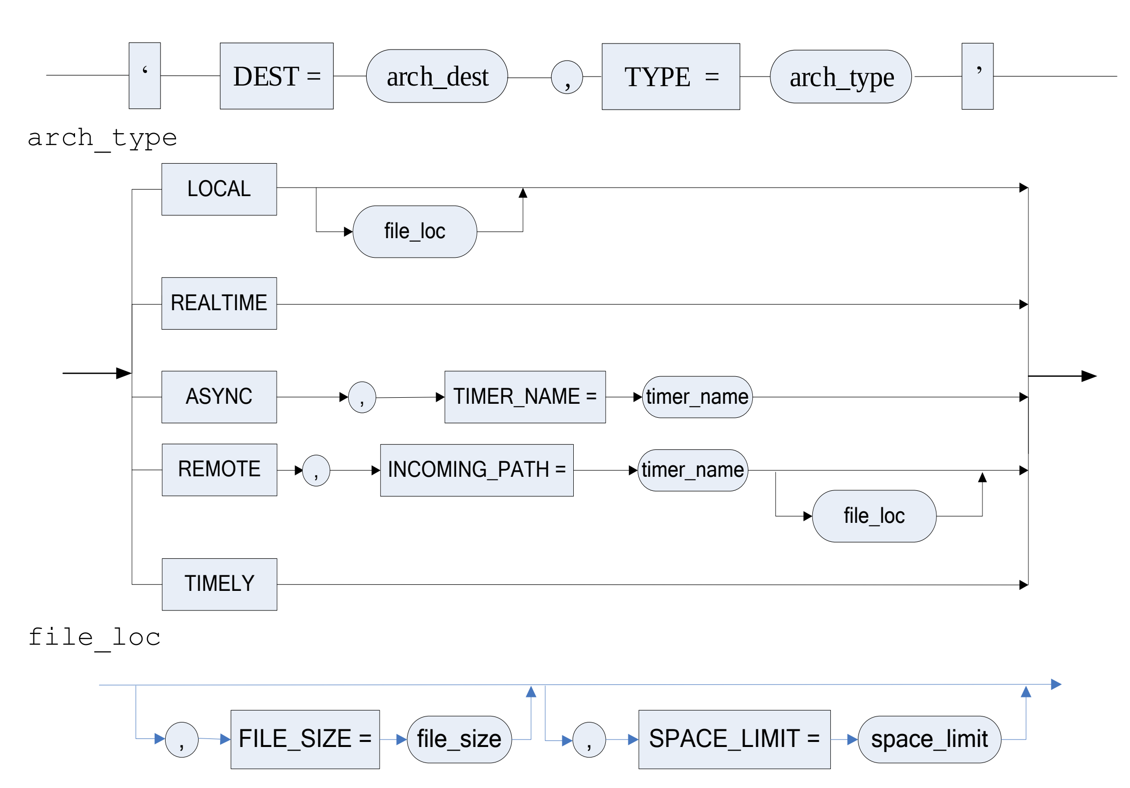
图例

ALTER DATABASE RESIZE LOGFILE filepath TO filesize ;
ADD LOGFILE file_item ,
ADD NODE LOGFILE file_item , file_item ,
RENAME LOGFILE filepath , TO filepath ,
OPEN FORCE
MOUNT
SUSPEND
NORMAL
PRIMARY
STANDBY
ARCHIVELOG
NOARCHIVELOG
ADD
DELETE ARCHIVELOG archive_configure_clause
MODIFY
ARCHIVELOG CURRENT

file_item



archive configure_clause



语句功能

供具有 DBA 权限的用户修改数据库。

使用说明

1. 归档的配置也可以通过 dm.ini 参数 ARCH_INI 和归档配置文件 dmarch.ini 进行，可参看《DM8 系统管理员手册》，SQL 语句提供了在 DM 服务器运行时对归档配置进行动态修改的手段，通过 SQL 语句修改成功后会将相关配置写入 dmarch.ini 中；
2. 修改日志文件大小时，只能增加文件的大小，否则失败；
3. ADD NODE LOGFILE 用于 DMDSC 集群扩展节点时使用；
4. 只有 MOUNT 状态 NORMAL 模式下才能启用或关闭归档，添加、修改、删除归档，重命名日志文件；
5. 归档模式下，不允许删除本地归档；
6. ARCHIVELOG CURRENT 把新生成的，还未归档的联机日志都进行归档。

举例说明

假设数据库 BOOKSHOP 页面大小为 8K，数据文件存放路径为 C:\DMDBMS\data。

例 1 给数据库增加一个日志文件 C:\DMDBMS\data\dmlog_0.log，其大小为 200M。

```
ALTER DATABASE ADD LOGFILE 'C:\DMDBMS\data\dmlog_0.log' SIZE 200;
```

例 2 扩展数据库中的日志文件 C:\DMDBMS\data\dmlog_0.log，使其大小增大为 300M。

```
ALTER DATABASE RESIZE LOGFILE 'C:\DMDBMS\data\dmlog_0.log' TO 300;
```

例 3 设置数据库状态为 MOUNT。

```
ALTER DATABASE MOUNT;
```

例 4 设置数据库状态为 OPEN。

```
ALTER DATABASE OPEN;
```

例 5 设置数据库状态为 SUSPEND。

```
ALTER DATABASE SUSPEND;
```

例 6 重命名日志文件 C:\DMDBMS\data\dmlog_0.log 为 d:\dmlog_1.log。

```
ALTER DATABASE MOUNT;
ALTER DATABASE RENAME LOGFILE 'C:\DMDBMS\data\dmlog_0.log' TO 'd:\dmlog_1.log';
ALTER DATABASE OPEN;
```

例 7 设置数据库模式为 PRIMARY。

```
ALTER DATABASE MOUNT;
ALTER DATABASE PRIMARY;
ALTER DATABASE OPEN FORCE;
```

例 8 设置数据库模式为 STANDBY。

```
ALTER DATABASE MOUNT;
ALTER DATABASE STANDBY;
ALTER DATABASE OPEN FORCE;
```

例 9 设置数据库模式为 NORMAL。

```
ALTER DATABASE MOUNT;
ALTER DATABASE NORMAL;
ALTER DATABASE OPEN;
```

例 10 设置数据归档模式为非归档。

```
ALTER DATABASE MOUNT;
ALTER DATABASE NOARCHIVELOG;
```

例 11 设置数据库归档模式为归档。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ARCHIVELOG;
```

例 12 增加本地归档配置，归档目录为 c:\arch_local，文件大小为 128MB，空间限制为 1024MB。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ADD ARCHIVELOG 'DEST = c:\arch_local, TYPE = local, FILE_SIZE = 128, SPACE_LIMIT = 1024';
```

例 13 增加一个实时归档配置，远程服务实例名为 realtime，需事先配置 mail。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ADD ARCHIVELOG 'DEST = realtime, TYPE = REALTIME';
```

例 14 增加一个异步归档配置，远程服务实例名为 asyn，定时器名为 timer1，需事先配置好 mail 和 timer。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ADD ARCHIVELOG 'DEST = asyn, TYPE = ASYNC, TIMER_NAME = timer1';
```

3.2管理用户

3.2.1 用户定义语句

在数据库中创建新的用户，DM 中直接用 USER 与数据库服务器建立连接。

语法格式

```
CREATE USER <用户名> IDENTIFIED <身份验证模式> [PASSWORD_POLICY <口令策略>][<锁定子句>][<存储加密密钥>][<空间限制子句>][<只读标志>][<资源限制子句>][<允许 IP 子句>][<
```

```
禁止 IP 子句>][<允许时间子句>][<禁止时间子句>][<TABLESPACE 子句>][<INDEX_TABLESPACE 子句>]
    <身份验证模式> ::= <数据库身份验证模式>|<外部身份验证模式>
    <数据库身份验证模式> ::= BY <口令> [<散列选项>]
    <散列选项> ::= HASH WITH [<密码引擎名>.]<散列算法> [<加盐选项>]
    <散列算法> ::= <MD5 | SHA1 | SHA224 | SHA256 | SHA384 | SHA512>
    <加盐选项> ::= [NO] SALT
    <外部身份验证模式> ::= EXTERNALLY | EXTERNALLY AS <用户 DN>
    <口令策略> ::= 口令策略项的任意组合
    <锁定子句> ::= ACCOUNT LOCK | ACCOUNT UNLOCK
    <存储加密密钥> ::= ENCRYPT BY <口令>
    <空间限制子句> ::= DISKSPACE LIMIT <空间大小>| DISKSPACE UNLIMITED
    <只读标志> ::= READ ONLY | NOT READ ONLY
    <资源限制子句> ::= LIMIT <资源设置项>{,<资源设置项>}
    <资源设置项> ::= SESSION_PER_USER <参数设置>|
                    CONNECT_IDLE_TIME <参数设置>|
                    CONNECT_TIME <参数设置>|
                    CPU_PER_CALL <参数设置>|
                    CPU_PER_SESSION <参数设置>|
                    MEM_SPACE <参数设置>|
                    READ_PER_CALL <参数设置>|
                    READ_PER_SESSION <参数设置>|
                    FAILED_LOGIN_ATTEMPS <参数设置>|
                    PASSWORD_LIFE_TIME <参数设置>|
                    PASSWORD_REUSE_TIME <参数设置>|
                    PASSWORD_REUSE_MAX <参数设置>|
                    PASSWORD_LOCK_TIME <参数设置>|
                    PASSWORD_GRACE_TIME <参数设置>
    <参数设置> ::=<参数值>| UNLIMITED
    <允许 IP 子句> ::= ALLOW_IP <IP 项>{,<IP 项>}
    <禁止 IP 子句> ::= NOT_ALLOW_IP <IP 项>{,<IP 项>}
    <IP 项> ::= <具体 IP>|<网段>
    <允许时间子句> ::= ALLOW_DATETIME <时间项>{,<时间项>}
    <禁止时间子句> ::= NOT_ALLOW_DATETIME <时间项>{,<时间项>}
    <时间项> ::= <具体时间段> | <规则时间段>
    <具体时间段> ::= <具体日期> <具体时间> TO <具体日期> <具体时间>
    <规则时间段> ::= <规则时间标志> <具体时间> TO <规则时间标志> <具体时间>
    <规则时间标志> ::= MON | TUE | WED | THURS | FRI | SAT | SUN
    <TABLESPACE 子句> ::= DEFAULT TABLESPACE <表空间名>
    <INDEX_TABLESPACE 子句> ::= DEFAULT INDEX TABLESPACE <表空间名>
```

参数

1． <用户名> 指明要创建的用户名称，用户名称最大长度 128 字节；
2． <参数设置>用于限制用户对 DM 数据库服务器系统资源的使用；
3． 系统在创建用户时，必须指定一种身份验证模式：<数据库身份验证模式>或者<外

部身份验证模式>，外部身份验证模式支持基于操作系统(OS)的身份验证、LDAP 身份验证和 KERBEROS 身份验证，具体请参考《DM8 安全管理》2.3 节；

4. <口令策略>可以为以下值，或其任何组合：

0 无策略；

1 禁止与用户名相同；

2 口令长度不小于 9；

4 至少包含一个大写字母(A-Z)；

8 至少包含一个数字(0-9)；

16 至少包含一个标点符号(英文输入法状态下，除" 和空格外的所有符号)。

若为其他数字，则表示以上设置值的和，如 3＝1＋2，表示同时启用第 1 项和第 2 项策略。当设置为 0 时，表示设置口令没有限制，但总长度不得超过 48 个字节。另外，若不指定该项，则默认采用系统配置文件中 PWD_POLICY 所设值。

5. 存储加密密钥用于与半透明加密配合使用，缺省情况下系统自动生成一个密钥，半透明加密时用户仅能查看到自己插入的数据；

6. 空间限制子句用于限制用户使用的最大存储空间，以 M 为单位，取值范围为 1 到 1048576，关键字 UNLIMITED 表示无限制；

7. 只读标志表示该登录是否只能对数据库进行只读操作，默认为可读写；

8. 资源设置项的各参数设置说明见下表：

表 3.2.1 资源设置项说明

资源设置项	说明	最大值	最小值	缺省值
SESSION_PER_USER	在一个实例中，一个用户可以同时拥有的会话数量	32768	1	系统所能提供的最大值
CONNECT_TIME	一个会话连接、访问和操作数据库服务器的时间上限（单位：1 分钟）	1440（1 天）	1	无限制
CONNECT_IDLE_TIME	会话最大空闲时间（单位：1 分钟）	1440（1 天）	1	无限制
FAILED_LOGIN_ATTEMPS	将引起一个账户被锁定的连续注册失败的次数	100	1	3
CPU_PER_SESSION	一个会话允许使用的 CPU 时间上限（单位：秒）	31536000（365 天）	1	无限制
CPU_PER_CALL	用户的一个请求能够使用的 CPU 时间上限（单位：秒）	86400（1 天）	1	无限制
READ_PER_SESSION	会话能够读取的总数据页数上限	2147483646	1	无限制
READ_PER_CALL	每个请求能够读取的数据页数	2147483646	1	无限制
MEM_SPACE	会话占有的私有内存空间上限（单位：MB）	2147483647	1	无限制
PASSWORD_LIFE_TIME	一个口令在其终止前可以使用的天数	365	1	无限制
PASSWORD_REUSE_TIME	一个口令在可以重新使用前必须经过的天数	365	1	无限制
PASSWORD_REUSE_MAX	一个口令在可以重新使用前必须改变的次数	32768	1	无限制
PASSWORD_LOCK_TIME	如果超过 FAILED_LOGIN_ATTEMPS 设置值，一个账户将被锁定的分钟数	1440（1 天）	1	1
PASSWORD_GRACE_TIME	以天为单位的口令过期宽限时间	30	1	10

9. 允许 IP 和禁止 IP 用于控制此登录是否可以从某个 IP 访问数据库，其中禁止 IP 优先。在设置 IP 时，可以利用*来设置网段，如 192.168.0.*。设置的允许和禁止 IP 需要用双引号括起来；

10. 允许时间段和禁止时间段用于控制此登录是否可以在某个时间段访问数据库，其中

禁止时间段优先。设置的时间段中的日期和时间要分别用双引号括起来。在设置时间段时，有两种方式：

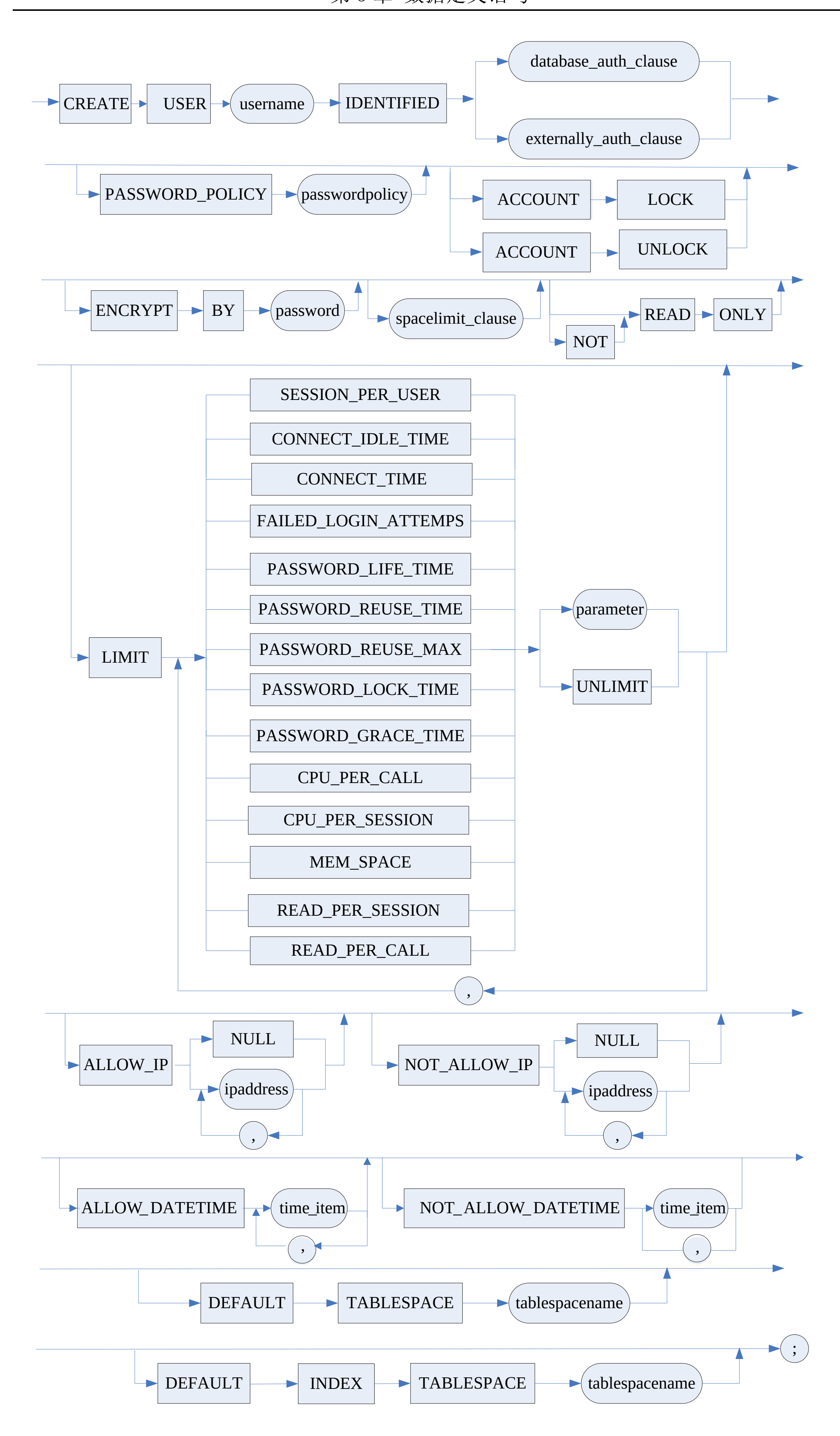
1） 具体时间段，如 2016 年 1 月 1 日 8：30 至 2006 年 2 月 1 日 17：00；

2） 规则时间段，如 每周一 8：30 至 每周五 17：00。

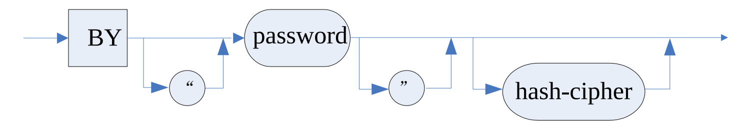
11．口令策略、允许 IP、禁止 IP、允许时间段、禁止时间段和外部身份验证功能只在安全版本中提供；

12．用户默认表空间和索引默认表空间不能使用 SYSTEM、RLOG、ROLL、TEMP 表空间。

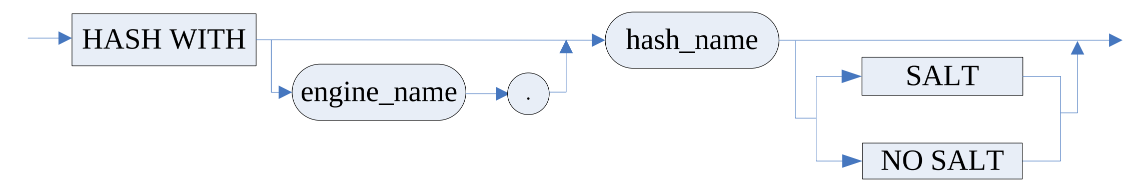
图例


CREATE
USER
username
IDENTIFIED
database_auth_clause
externally_auth_clause
PASSWORD_POLICY
passwordpolicy
ACCOUNT
LOCK
ACCOUNT
UNLOCK
ENCRYPT
BY
password
spacelimit_clause
NOT
READ
ONLY
LIMIT
SESSION_PER_USER
CONNECT_IDLE_TIME
CONNECT_TIME
FAILED_LOGIN_ATTEMPS
PASSWORD_LIFE_TIME
PASSWORD_REUSE_TIME
PASSWORD_REUSE_MAX
PASSWORD_LOCK_TIME
PASSWORD_GRACE_TIME
CPU_PER_CALL
CPU_PER_SESSION
MEM_SPACE
READ_PER_SESSION
READ_PER_CALL
parameter
UNLIMIT
,
ALLOW_IP
NULL
ipaddress
,
NOT_ALLOW_IP
NULL
ipaddress
,
ALLOW_DATETIME
time_item
,
NOT_ALLOW_DATETIME
time_item
,
DEFAULT
TABLESPACE
tablespacename
DEFAULT
INDEX
TABLESPACE
tablespacename
;

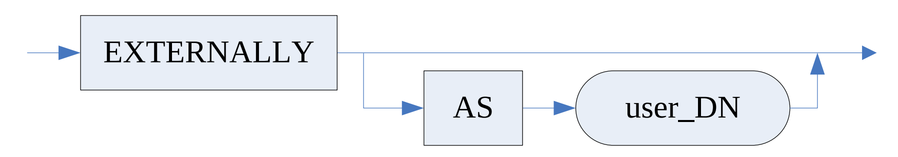
database_auth_clause



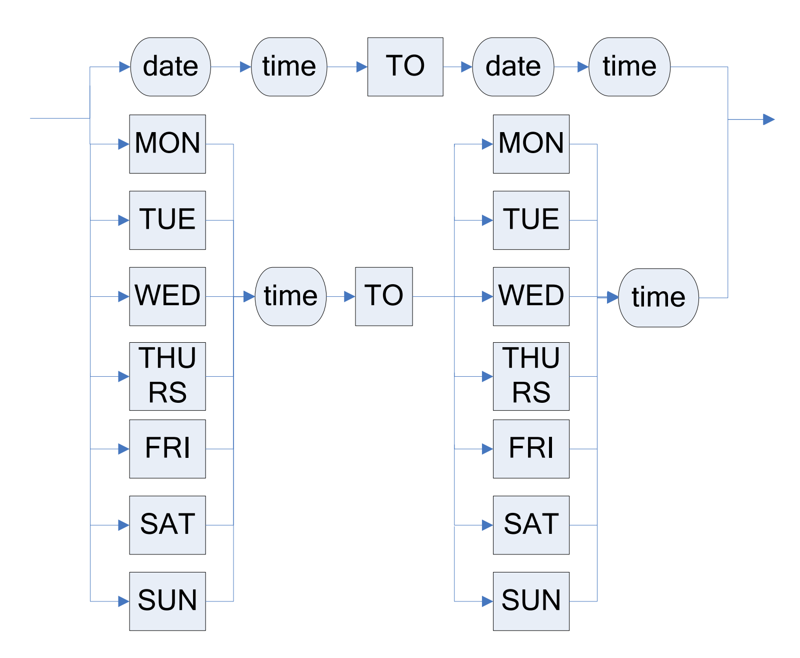
hash_cipher



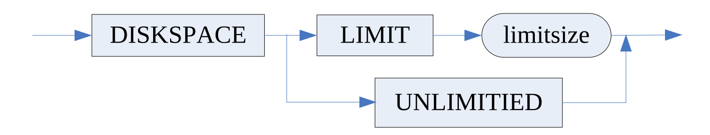
externally_auth_clause



time_item



spacelimit_clause



语句功能

创建新的用户。

使用说明

1． 用户名在服务器中必须唯一；

2． 系统为一个用户存储的信息主要有：用户名、口令、资源限制；

3． 用户口令以密文形式存储；

4． 如果没有指定用户默认表空间，则系统指定 MAIN 表空间为用户的默认表空间；

5． 如果没有指定用户默认索引表空间，则 HUGE 表的索引缺省存储在用户的默认表空间中，普通表的索引缺省存储在表的聚集索引所在的表空间中。临时表的索引永远在 TEMP 表空间；

6． 系统预先设置了三个用户，分别为 SYSDBA、SYSAUDITOR 和 SYSSSO，其中 SYSDBA 具备 DBA 角色，SYSAUDITOR 具备 DB_AUDIT_ADMIN 角色，而 SYSSSO 具备 DB_POLICY_ADMIN 系统角色；

7． 不指定<散列选项>时，默认使用 SHA512 算法；

8． DM 提供数据库身份验证模式和外部身份验证模式来保护对数据库访问的安全。数据库身份验证模式需要利用数据库口令；外部身份验证模式支持基于操作系统(OS)的身份验证、LDAP 身份验证和 KERBEROS 身份验证，关于外部身份验证模式的使用具体请参考《DM8 安全管理》。

举例说明

例 1 创建用户名为 BOOKSHOP_USER、口令为 BOOKSHOP_PASSWORD、会话超时为 30 分钟的用户。

```
CREATE USER BOOKSHOP_USER IDENTIFIED BY BOOKSHOP_PASSWORD LIMIT CONNECT_TIME 3;
```

例 2 创建用户名为 BOOKSHOP_OS_USER、基于操作系统身份验证的用户。

```
CREATE USER BOOKSHOP_OS_USER IDENTIFIED EXTERNALLY;
```

3.2.2 修改用户语句

修改数据库中的用户。

语法格式

```
ALTER USER <用户名> [IDENTIFIED <身份验证模式>] [PASSWORD_POLICY <口令策略>] [<锁定子句>] [<存储加密密钥>] [<空间限制子句>] [<只读标志>][<资源限制子句>][<允许 IP 子句>][<禁止 IP 子句>][<允许时间子句>][<禁止时间子句>][<TABLESPACE 子句>][<INDEX_TABLESPACE 子句>][<SCHEMA 子句>];
<身份验证模式> ::= <数据库身份验证模式>|<外部身份验证模式>
<数据库身份验证模式> ::= BY <口令> [<散列选项>]
<散列选项> ::= HASH WITH [<密码引擎名>.]<散列算法> [<加盐选项>]
<散列算法> ::= <MD5 | SHA1 | SHA224 | SHA256 | SHA384 | SHA512>
<加盐选项> ::= [NO] SALT
<外部身份验证模式> ::= EXTERNALLY | EXTERNALLY AS <用户 DN>
<口令策略> ::= 口令策略项的任意组合
<锁定子句> ::= ACCOUNT LOCK | ACCOUNT UNLOCK
<存储加密密钥> ::= ENCRYPT BY <口令>
<空间限制子句> ::= DISKSPACE LIMIT <空间大小>| DISKSPACE UNLIMITED
<只读标志> ::= READ ONLY | NOT READ ONLY
<资源限制子句> ::= LIMIT <资源设置项>{,<资源设置项>}
<资源设置项> ::= SESSION_PER_USER <参数设置>|
                 CONNECT_IDLE_TIME <参数设置>|
```

```
                    CONNECT_TIME <参数设置>|
                    CPU_PER_CALL <参数设置> |
                    CPU_PER_SESSION <参数设置> |
                    MEM_SPACE <参数设置> |
                    READ_PER_CALL <参数设置> |
                    READ_PER_SESSION <参数设置> |
                    FAILED_LOGIN_ATTEMPS <参数设置>|
                    PASSWORD_LIFE_TIME <参数设置>|
                    PASSWORD_REUSE_TIME <参数设置>|
                    PASSWORD_REUSE_MAX <参数设置>|
                    PASSWORD_LOCK_TIME <参数设置>|
                    PASSWORD_GRACE_TIME <参数设置>
<参数设置> ::=<参数值>| UNLIMITED
<允许 IP 子句> ::= ALLOW_IP <IP 项>{,<IP 项>}
<禁止 IP 子句> ::= NOT_ALLOW_IP <IP 项>{,<IP 项>}
<IP 项> ::= <具体 IP>|<网段>
<允许时间子句> ::= ALLOW_DATETIME <时间项>{,<时间项>}
<禁止时间子句> ::= NOT_ALLOW_DATETIME <时间项>{,<时间项>}
<时间项> ::= <具体时间段> | <规则时间段>
<具体时间段> ::= <具体日期> <具体时间> TO <具体日期> <具体时间>
<规则时间段> ::= <规则时间标志> <具体时间> TO <规则时间标志> <具体时间>
<规则时间标志> ::= MON | TUE | WED | THURS | FRI | SAT | SUN
<TABLESPACE 子句> ::=DEFAULT TABLESPACE <表空间名>
<INDEX_TABLESPACE 子句> ::= DEFAULT INDEX TABLESPACE <表空间名>
<SCHEMA 子句> ::= ON SCHEMA <模式名>
```

参数

同用户定义语句的参数规定一样。

图例

参考用户定义语句的图例

语句功能

修改用户。

使用说明

1．每个用户均可修改自身的口令，SYSDBA 用户可强制修改所有其他用户的口令(在数据库验证方式下)；

2．只有具备 ALTER USER 权限的用户才能修改其身份验证模式、系统角色及资源限制项；

3．不论 dm.ini 的 DDL_AUTO_COMMIT 设置为自动提交还是非自动提交，ALTER USER 操作都会被自动提交；

4．修改用户口令时，口令策略应符合创建该用户时指定的口令策略；

5．不能修改系统固定用户的系统角色和资源限制项；

6．<SCHEMA 子句>用于设置用户的缺省模式；

7．其他参数的取值、意义与 CREATE USER 中的要求一样。

举例说明

例 修改用户 BOOKSHOP_USER，会话空闲期为无限制，最大连接数为 10。

```
    ALTER USER BOOKSHOP_USER LIMIT SESSION_PER_USER 10, CONNECT_IDLE_TIME
UNLIMITED;
```

3.2.3 用户删除语句

删除用户。

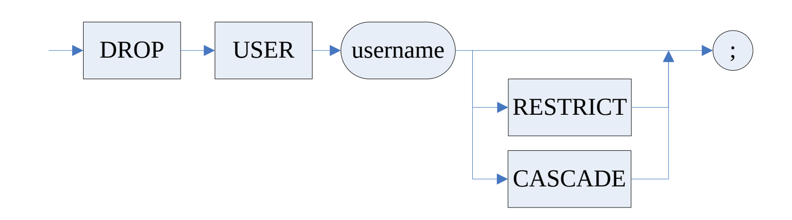
语法格式

```
DROP USER <用户名> [RESTRICT | CASCADE];
```

参数

<用户名> 指明被删除的用户。

图例



语句功能

删除指定用户。

使用说明

1. 系统自动创建的三个系统用户 SYSDBA、SYSAUDITOR 和 SYSSSO 不能被删除；
2. 具有 DROP USER 权限的用户即可进行删除用户操作；
3. 执行此语句将导致 DM 删除数据库中该用户建立的所有对象，且不可恢复。如果要保存这些实体，请参考 REVOKE 语句；
4. 如果未使用 CASCADE 选项，若该用户建立了数据库对象（如表、视图、过程或函数），或其他用户对象引用了该用户的对象，或在该用户的表上存在其它用户建立的视图，DM 将返回错误信息，而不删除此用户；
5. 如果使用了 CASCADE 选项，除数据库中该用户及其创建的所有对象被删除外，如果其他用户创建的表引用了该用户表上的主关键字或唯一关键字，或者在该表上创建了视图，DM 还将自动删除相应的引用完整性约束及视图依赖关系；
6. 正在使用中的用户可以被删除，删除后重登录或者做操作会报错。

举例说明

例 1 删除用户 BOOKSHOP_USER。

```
DROP USER BOOKSHOP_USER;
```

例 2 删除用户 BOOKSHOP_OS_USER。

```
DROP USER BOOKSHOP_OS_USER CASCADE;
```

3.3 管理模式

3.3.1 模式定义语句

模式定义语句创建一个架构，并且可以在概念上将其看作是包含表、视图和权限定义的对象。在 DM 中，一个用户可以创建多个模式，一个模式中的对象(表、视图)可以被多个用

户使用。

系统为每一个用户自动建立了一个与用户名同名的模式作为默认模式，用户还可以用模式定义语句建立其它模式。

语法格式

<模式定义子句 1> | <模式定义子句 2>

<模式定义子句 1> ::= CREATE SCHEMA <模式名> [AUTHORIZATION <用户名>][<DDL_GRANT 子句> {< DDL_GRANT 子句>}];

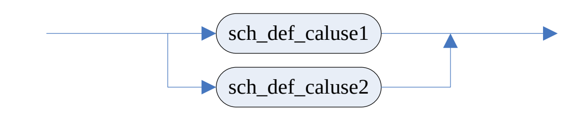
<模式定义子句 2> ::= CREATE SCHEMA AUTHORIZATION <用户名> [<DDL_GRANT 子句> {< DDL_GRANT 子句>}];

<DDL_GRANT 子句> ::= <基表定义> | <域定义>| <基表修改> | <索引定义> | <视图定义> | <序列定义> | <存储过程定义> | <存储函数定义> | <触发器定义> | <特权定义> | <全文索引定义> | <同义词定义> | <包定义> | <包体定义> | <类定义> | <类体定义> | <外部链接定义>] | <物化视图定义> | <物化视图日志定义> | <注释定义>

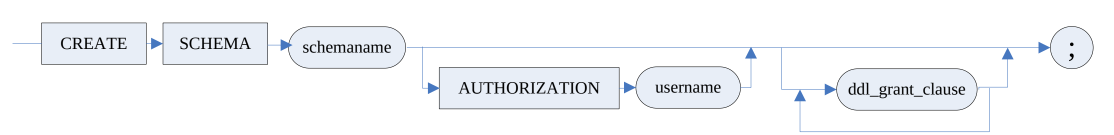
参数

1. <模式名> 指明要创建的模式的名字，最大长度 128 字节；
2. <基表定义> 建表语句；
3. <域定义> 域定义语句；
4. <基表修改> 基表修改语句；
5. <索引定义> 索引定义语句；
6. <视图定义> 建视图语句；
7. <序列定义> 建序列语句；
8. <存储过程定义> 存储过程定义语句；
9. <存储函数定义> 存储函数定义语句；
10. <触发器定义> 建触发器语句；
11. <特权定义> 授权语句；
12. <全文索引定义> 全文索引定义语句；
13. <同义词定义> 同义词定义语句；
14. <包定义> 包定义语句；
15. <包体定义> 包体定义语句；
16. <类定义> 类定义语句；
17. <类体定义> 类体定义语句；
18. <外部链接定义> 外部链接定义语句；
19. <物化视图定义> 物化视图定义语句；
20. <物化视图日志定义> 物化视图日志定义语句；
21. <注释定义> 注释定义语句。

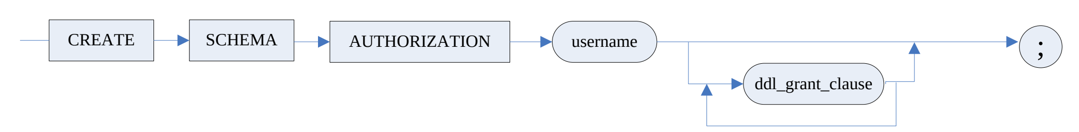
图例



sch_def_clause1



sch_def_clause2



ddl_grant_clause：略

语句功能

供具有 DBA 或 CREATE SCHEMA 权限的用户在指定数据库中定义模式。

使用说明

1．在创建新的模式时，如果已存在同名的模式，或当存在能够按名字不区分大小写匹配的同名用户时（此时认为模式名为该用户的默认模式），那么创建模式的操作会被跳过，而如果后续还有 DDL 子句，根据权限判断是否可在已存在模式上执行这些 DDL 操作；

2．AUTHORIZATION <用户名>标识了拥有该模式的用户；它是为其他用户创建模式时使用的；缺省拥有该模式的用户为 SYSDBA；

3．使用 sch_def_clause2 创建模式时，模式名与用户名相同；

4．使用该语句的用户必须具有 DBA 或 CREATE SCHEMA 权限；

5．DM 使用 DMSQL 程序模式执行创建模式语句，因此创建模式语句中的标识符不能使用系统的保留字；

6．定义模式时，用户可以用单条语句同时建多个表、视图，同时进行多项授权；

7．模式一旦定义，该用户所建基表、视图等均属该模式，其它用户访问该用户所建立的基表、视图等均需在表名、视图名前冠以模式名；而建表者访问自己当前模式所建表、视图时模式名可省；若没有指定当前模式，系统自动以当前用户名作为模式名；

8．模式定义语句中的基表修改子句只允许添加表约束；

9．模式定义语句中的索引定义子句不能定义聚集索引；

10．模式未定义之前，其它用户访问该用户所建的基表、视图等均需在表名前冠以建表者名；

11．模式定义语句不允许与其它 SQL 语句一起执行；

12．在 DIsql 中使用该语句必须以"/"结束。

举例说明

例 用户 SYSDBA 创建模式 SCHEMA1，建立的模式属于 SYSDBA。

```
CREATE SCHEMA SCHEMA1 AUTHORIZATION SYSDBA;
```

3.3.2 设置当前模式语句

设置当前模式。

语法格式

```
SET SCHEMA <模式名>;
```

图例



使用说明

只能设置成属于自己的模式。

举例说明

例 SYSDBA 用户将当前的模式从 SYSDBA 换到 SALES 模式。

```
SET SCHEMA SALES;
```

3.3.3 模式删除语句

在 DM 系统中，允许用户删除整个模式。

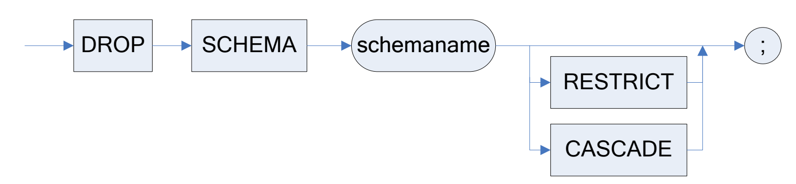
语法格式

```
DROP SCHEMA <模式名> [RESTRICT | CASCADE];
```

参数

<模式名> 指要删除的模式名。

图例



语句功能

供具有 DBA 角色的用户或该模式的拥有者删除模式。

使用说明

1. <模式名>必须是当前数据库中已经存在的模式；
2. 用该语句的用户必须具有 DBA 权限或是该模式的所有者；
3. 如果使用 RESTRICT 选项，只有当模式为空时删除才能成功，否则，当模式中存在数据库对象时则删除失败。默认选项为 RESTRICT 选项；
4. 如果使用 CASCADE 选项，则将整个模式、模式中的对象，以及与该模式相关的依赖关系都删除。

举例说明

例 以 SYSDBA 身份登录数据库后，删除 BOOKSHOP 库中模式 SCHEMA1。

```
DROP SCHEMA SCHEMA1 CASCADE;
```

3.4 管理表空间

3.4.1 表空间定义语句

创建表空间。

语法格式

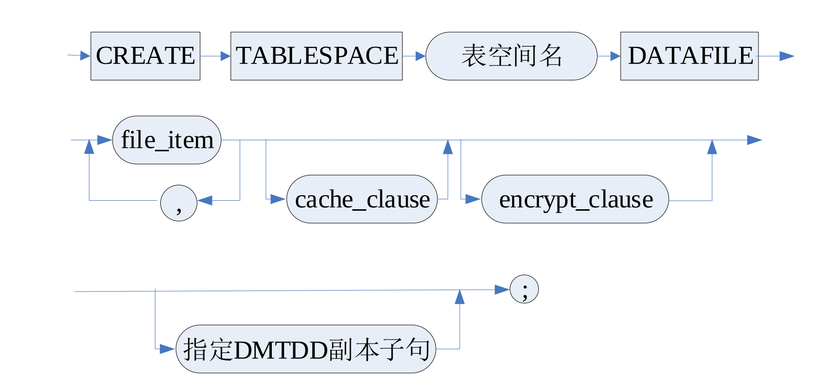
```
CREATE TABLESPACE <表空间名> <数据文件子句>[<数据页缓冲池子句>][<存储加密子句>][<指定 DMTDD 副本子句>]
<数据文件子句> ::= DATAFILE <文件说明项>{,<文件说明项>}
<文件说明项> ::= <文件路径> [ MIRROR <文件路径>] SIZE <文件大小>[<自动扩展子句>]
<自动扩展子句> ::= AUTOEXTEND <ON [<每次扩展大小子句>][<最大大小子句> |OFF>
<每次扩展大小子句> ::= NEXT <扩展大小>
<最大大小子句> ::= MAXSIZE <文件最大大小>
```

```
<数据页缓冲池子句> ::= CACHE = <缓冲池名>
<存储加密子句> ::= ENCRYPT WITH <加密算法> [[BY] <加密密码>]
<指定 DMTDD 副本子句> ::= [<指定副本数子句>][<副本策略子句>]
<指定副本数子句> ::= COPY <副本数>
<副本策略子句> ::= GREAT | MICRO
```

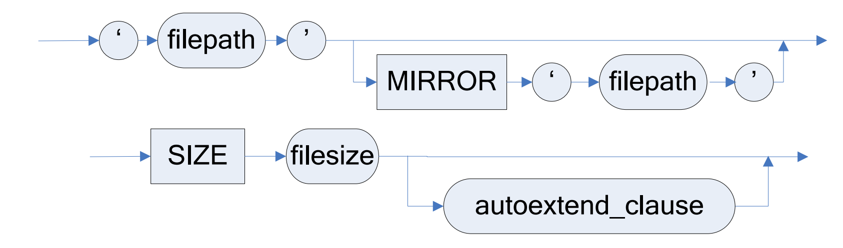
参数

1. <表空间名> 表空间的名称，最大长度 128 字节；
2. <文件路径> 指明新生成的数据文件在操作系统下的路径＋新数据文件名。数据文件的存放路径符合 DM 安装路径的规则，且该路径必须是已经存在的；
3. MIRROR 数据文件镜像，用于在数据文件出现损坏时替代数据文件进行服务；MIRROR 数据文件的<文件路径>必须是绝对路径。要使用数据文件镜像，必须在建库时开启页校验的参数 PAGE_CHECK；
4. <文件大小> 整数值，指明新增数据文件的大小(单位 MB)，取值范围 4096*页大小~2147483647*页大小；
5. <缓冲池名> 系统数据页缓冲池名 NORMAL 或 KEEP。缓冲池名 KEEP 是 DM 的保留关键字，使用时必须加双引号；
6. <加密算法> 可通过查看动态视图 V$CIPHERS 获取算法名；
7. <加密密码> 最大长度 128 字节，若未指定，由 DM 随机生成；
8. <副本数> 表空间文件在 DMTDD 中的副本数，默认为 DMDFS.INI 中的 DFS_COPY_NUM 的值；
9. <副本策略子句> 指定管理 DMTDD 副本的区块：宏区(GREAT)或是微区(MICRO)。

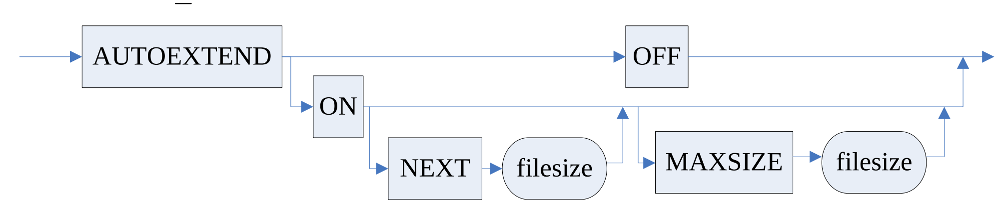
图例



file_item



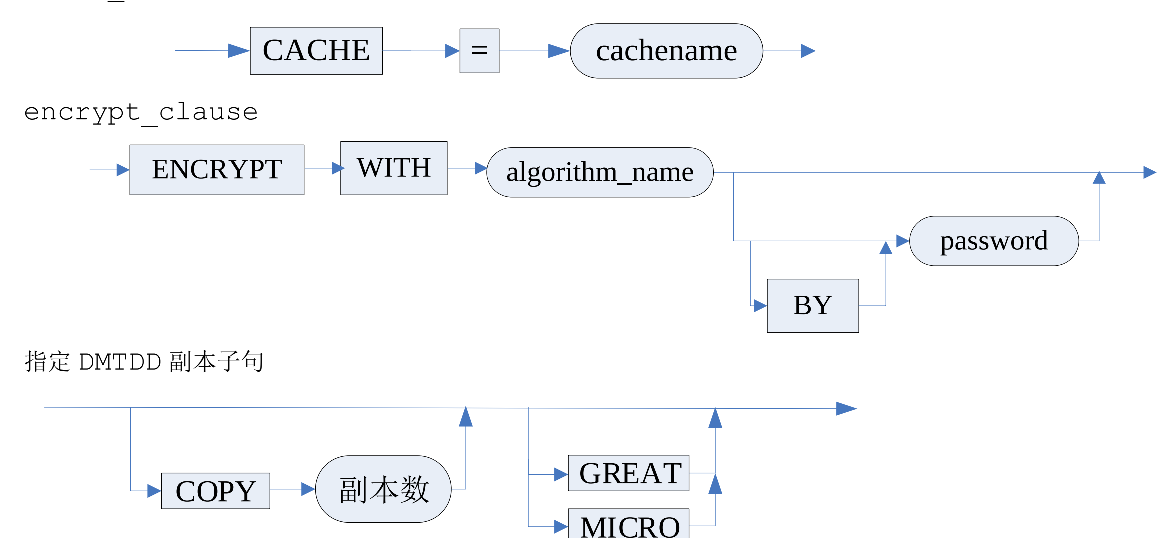
autoextend_clause



cache_clause

encrypt_clause

指定 DMTDD 副本子句



语句功能

供具有权限的用户创建表空间。

使用说明

1． 表空间名在数据库中必须唯一；
2． 一个表空间中，数据文件和镜像文件一起不能超过 256 个；
3． 如果全库已经加密，就不再支持表空间加密；
4． SYSTEM 表空间不允许关闭自动扩展，且不允许限制空间大小。

举例说明

例 以 SYSDBA 身份登录数据库后，创建表空间 TS1，指定数据文件 TS1.dbf，大小 128M。

```
CREATE TABLESPACE TS1 DATAFILE 'd:\TS1.dbf' SIZE 128;
```

3.4.2 修改表空间语句

修改表空间。

语法格式

```
ALTER TABLESPACE <表空间名> [ONLINE | OFFLINE | CORRUPT|<表空间重命名子句> | <数据文件重命名子句>|<增加数据文件子句>|<修改文件大小子句>|<修改文件自动扩展子句>|<数据页缓冲池子句>
<表空间重命名子句> ::= RENAME TO <表空间名>
<数据文件重命名子句>::= RENAME DATAFILE <文件路径>{,<文件路径>} TO <文件路径>{,<文件路径>}
<增加数据文件子句> ::= ADD <数据文件子句>
<数据文件子句>见上一节表空间定义语句
<修改文件大小子句> ::= RESIZE DATAFILE <文件路径> TO <文件大小>
<修改文件自动扩展子句> ::= DATAFILE <文件路径>{,<文件路径>}[<自动扩展子句>]
<自动扩展子句> ::= 见 3.4.1 节说明
<数据页缓冲池子句> ::= 见 3.4.1 节说明
```

参数

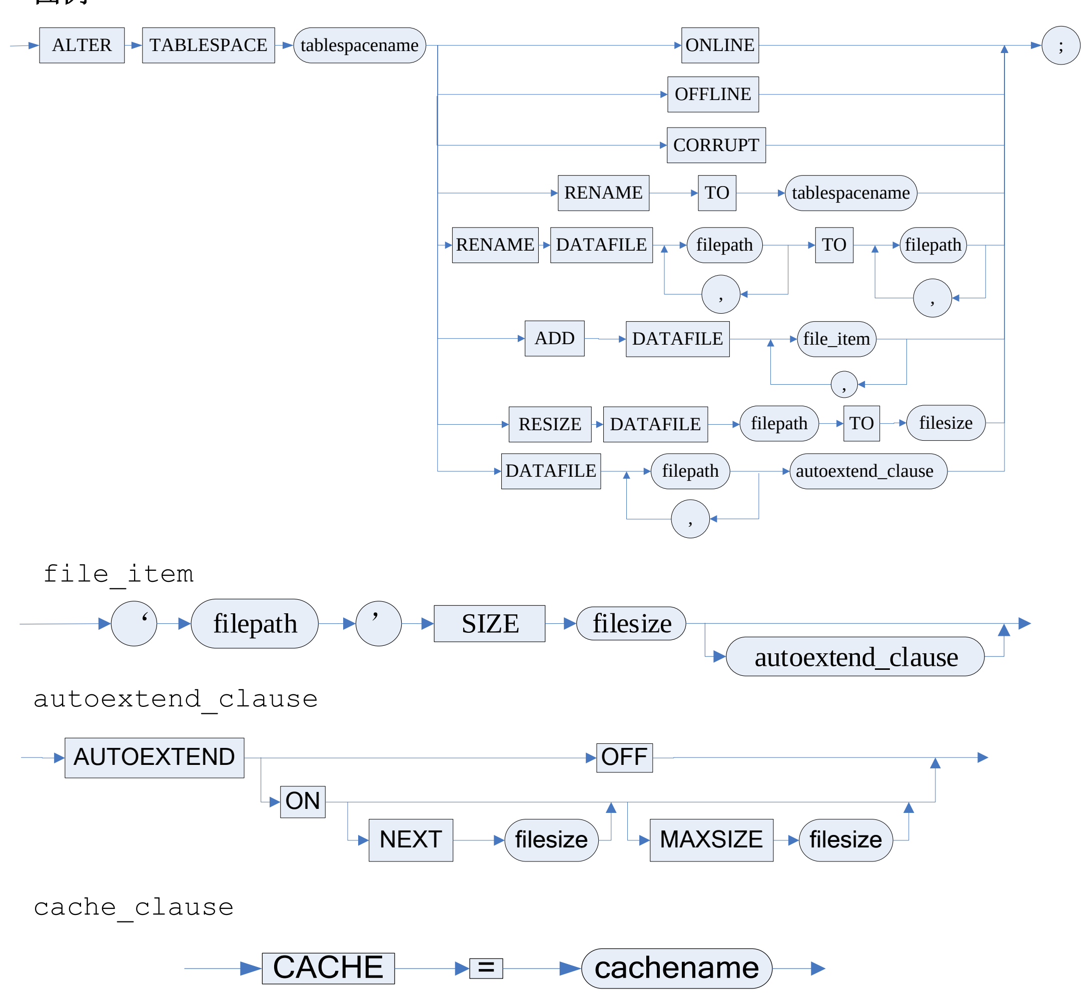
1． <表空间名> 表空间的名称；

2．<文件路径> 指明数据文件在操作系统下的路径＋新数据文件名。数据文件的存放路径符合 DM 安装路径的规则，且该路径必须是已经存在的；

3．<文件大小> 整数值，指明新增数据文件的大小(单位 MB)；

4．<缓冲池名> 系统数据页缓冲池名 NORMAL 或 KEEP。

图例



语句功能

供具有权限的用户修改表空间。

使用说明

1．不论 dm.ini 的 DDL_AUTO_COMMIT 设置为自动提交还是非自动提交，ALTER TABLESPACE 操作都会被自动提交；

2．修改表空间数据文件大小时，其大小必须大于自身大小；

3．SYSTEM 表空间不允许关闭自动扩展，且不允许限制空间大小；

4．如果表空间有未提交事务时，表空间不能修改为 OFFLINE 状态；

5．重命名表空间数据文件时，表空间必须处于 OFFLINE 状态，修改成功后再将表空间修改为 ONLINE 状态；

6．表空间如果发生损坏（表空间还原失败，或者数据文件丢失或损坏）的情况下，允许将表空间切换为 CORRUPT 状态，并删除损坏的表空间，如果表空间上定义有对象，需要先将所有对象删除，再删除表空间。

举例说明

例 1 将表空间 TS1 名字修改为 TS2。

```
ALTER TABLESPACE TS1 RENAME TO TS2;
```

例 2 增加一个路径为 d:\TS1_1.dbf，大小为 128M 的数据文件到表空间 TS1。

```
ALTER TABLESPACE TS1 ADD DATAFILE  'd:\TS1_1.dbf' SIZE 128;
```

例 3 修改表空间 TS1 中数据文件 d:\TS1.dbf 的大小为 200M。

```
ALTER TABLESPACE TS1 RESIZE DATAFILE  'd:\TS1.dbf' TO 200;
```

例 4 重命名表空间 TS1 的数据文件 d:\TS1.dbf 为 e:\TS1_0.dbf。

```
ALTER TABLESPACE TS1 OFFLINE;
ALTER TABLESPACE TS1 RENAME DATAFILE 'd:\TS1.dbf' TO 'e:\TS1_0.dbf';
ALTER TABLESPACE TS1 ONLINE;
```

例 5 修改表空间 TS1 的数据文件 d:\TS1.dbf 自动扩展属性为每次扩展 10M，最大文件大小为 1G。

```
ALTER TABLESPACE TS1 DATAFILE  'd:\TS1.dbf' AUTOEXTEND ON NEXT 10 MAXSIZE 1000;
```

例 6 修改表空间 TS1 缓冲池名字为 KEEP。

```
ALTER TABLESPACE TS1 CACHE="KEEP";
```

例 7 修改表空间为 CORRUPT 状态，注意只有在表空间处于 OFFLINE 状态或表空间损坏的情况下才允许使用。

```
ALTER TABLESPACE TS1 CORRUPT;
```

3.4.3 表空间删除语句

删除表空间。

语法格式

```
DROP TABLESPACE <表空间名>
```

参数

<表空间名>　所要删除的表空间的名称。

图例



语句功能

供具有权限的用户删除表空间。

使用说明

1． SYSTEM、RLOG、ROLL 和 TEMP 表空间不允许删除；

2． 系统处于 SUSPEND 或 MOUNT 状态时不允许删除表空间，系统只有处于 OPEN 状态下才允许删除表空间。

举例说明

例 以 SYSDBA 身份登录数据库后，删除表空间 TS1。

```
DROP TABLESPACE TS1;
```

3.4.4 表空间失效文件检查

表空间恢复失效文件的检查。

语法格式

```
SP_FILE_SYS_CHECK ();
```

语句功能

在 LINUX 操作系统下，检查是否有数据文件被删除。

使用说明

该过程只在 LINUX 下有效。

举例说明

```
SP_FILE_SYS_CHECK ();
```

3.4.5 表空间失效文件恢复准备

表空间恢复失效文件的准备。

语法格式

```
SP_TABLESPACE_PREPARE_RECOVER(<表空间名>);
```

语句功能

在 LINUX 操作系统下，如果出现了正在使用数据文件被删除的情况，该过程完成失效文件恢复的准备工作。

使用说明

该过程只在 LINUX 下有效。

举例说明

```
SP_TABLESPACE_PREPARE_RECOVER('MAIN');
```

3.4.6 表空间失效文件恢复

表空间失效文件的恢复。

语法格式

```
SP_TABLESPACE_RECOVER(<表空间名>);
```

语句功能

在 LINUX 操作系统下，如果出现了正在使用数据文件被删除的情况，在调用了恢复准备的 SP_TABLESPACE_PREPARE_RECOVER 及在 OS 系统内完成了数据文件的复制后，调用该过程完成文件的恢复工作。表空间失效文件恢复的详细步骤可参考《DM8 系统管理员手册》

使用说明

1. 该过程只在 LINUX 下有效；
2. 在 SP_TABLESPACE_PREPARE_RECOVER 及在 OS 系统内完成了数据文件的复制后调用。

举例说明

```
SP_TABLESPACE_RECOVER('MAIN');
```

3.5 管理 HTS 表空间

创建 HUGE 表之前，应要先创建一个 HUGE 表空间（HTS）。如果不创建，只能使用系统 HUGE 表空间 HMAIN。

3.5.1 创建 HTS 表空间

语法格式

```
CREATE HUGE TABLESPACE <表空间名> PATH <表空间路径>;
```

参数

1. <表空间名> 表空间的名称，表空间名称最大长度 128 字节；
2. <表空间路径> 指明新生成的表空间在操作系统下的路径。

图例



语句功能

供具有权限的用户创建 HTS 表空间。

使用说明

1. 表空间名在服务器中必须唯一；
2. HTS 表空间不支持创建 MIRROR 镜像文件。

举例说明

例 创建表空间 HTS_NAME。

```
CREATE HUGE TABLESPACE HTS_NAME PATH 'e:\HTSSPACE';
```

3.5.2 修改 HTS 表空间

语法格式

```
ALTER HUGE TABLESPACE <表空间名> ADD PATH <路径>;
```

参数

1. <表空间名> 要修改的 HTS 表空间的名称；
2. <表空间路径> 指明要为 HTS 表空间添加的路径。

图例



语句功能

供具有权限的用户修改 HTS 表空间。

使用说明

一个 HTS 表空间最多支持 255 个路径。

举例说明

例 为表空间 HTS_NAME 添加路径。

```
ALTER HUGE TABLESPACE HTS_NAME ADD PATH 'e:\HTSPATH';
```

3.5.3 删除 HTS 表空间

删除 HTS 表空间。

语法格式

```
DROP HUGE TABLESPACE <表空间名>
```

参数

<表空间名> 所要删除的 HTS 表空间的名称。

图例



语句功能

供具有权限的用户删除 HTS 表空间。

使用说明

该表空间必须未被使用才能被删除。

举例说明

例 以 SYSDBA 身份登录数据库后，删除 HTS 表空间 HTS_NAME。

```
DROP HUGE TABLESPACE HTS_NAME;
```

3.6 管理表

3.6.1 表定义语句

用户数据库建立后，就可以定义基表来保存用户数据的结构。DM 数据库的表可以分为两类，分别为数据库内部表和外部表，数据库内部表由数据库管理系统自行组织管理，而外部表在数据库的外部组织，是操作系统文件。其中内部表包括：数据库基表、HUGE 表和水平分区表。手册中如无明确说明均指数据库基表。下面分别对这各种表的创建与使用进行详细描述。

3.6.1.1 定义数据库基表

用户数据库建立后，就可以定义基表来保存用户数据的结构。需指定如下信息：

1． 表名、表所属的模式名；

2． 列定义；

3． 完整性约束。

语法格式

```
CREATE [[GLOBAL] TEMPORARY] TABLE <表名定义> <表结构定义>;
<表名定义> ::= [<模式名>.] <表名>
<表结构定义>::=<表结构定义 1> | <表结构定义 2>
<表结构定义 1>::= (<列定义> {,<列定义>} [,<表级约束定义>{,<表级约束定义>}]) [ON COMMIT <DELETE | PRESERVE> ROWS] [<空间限制子句>] [<STORAGE 子句>][<压缩子句>][<高级日志子句>] [<add_log 子句>] [<DISTRIBUTE 子句>]
<表结构定义 2>::= [ON COMMIT <DELETE | PRESERVE> ROWS] [<空间限制子句>] [<STORAGE 子句>] [<压缩子句>]AS <不带 INTO 的 SELECT 语句>[<add_log 子句>] [<DISTRIBUTE 子句>];

<列定义> ::= <不同类型列定义> [<列定义子句>] [<STORAGE 子句>][<存储加密子句>]
<不同类型列定义> ::=<普通列定义>|<虚拟列列定义>
<普通列定义>::= <列名> <数据类型>
<虚拟列列定义>::=<列名>[<数据类型>] [GENERATED ALWAYS]AS (<虚拟列定义>) [VIRTUAL]
```

```
[VISIBLE]
    <列定义子句> ::=
        DEFAULT <列缺省值表达式> |
        <IDENTITY 子句> |
        <列级约束定义> |
        DEFAULT <列缺省值表达式> <列级约束定义> |
        <IDENTITY 子句> <列级约束定义> |
        <列级约束定义> DEFAULT <列缺省值表达式> |
        <列级约束定义> <IDENTITY 子句>
    <IDENTITY 子句> ::= IDENTITY [(<种子>,<增量>)]
    <列级约束定义> ::= <列级完整性约束>{,<列级完整性约束>}
    <列级完整性约束> ::= [CONSTRAINT <约束名>] < column_constraint_action>[<失效生效选项>]
    <column_constraint_action>::=
        [NOT] NULL |
        <唯一性约束选项> [USING INDEX TABLESPACE {<表空间名> | DEFAULT}]|
        <引用约束> |
        CHECK (<检验条件>)|
        NOT VISIBLE
    <唯一性约束选项> ::= [PRIMARY KEY ]| [[NOT] CLUSTER PRIMARY KEY] |[CLUSTER [UNIQUE] KEY] | UNIQUE |
    <引用约束> ::= [FOREIGN KEY] REFERENCES [PENDANT] [<模式名>.]<表名>[(<列名>{[,<列名>]})] [MATCH <FULL|PARTIAL|SIMPLE>][<引用触发动作>] [WITH INDEX]
    <引用触发动作> ::=
        <UPDATE 规则> [<DELETE 规则>] |
        <DELETE 规则> [<UPDATE 规则>]
    <UPDATE 规则> ::= ON UPDATE <引用动作>
    <DELETE 规则> ::= ON DELETE <引用动作>
    <引用动作> ::= CASCADE | SET NULL | SET DEFAULT | NO ACTION
    <失效生效选项>::=ENABLE | DISABLE

    <STORAGE 子句> ::= STORAGE(<STORAGE 项> {,<STORAGE 项>})
    <STORAGE 项> ::=
        [INITIAL <初始簇数目>] |
        [NEXT <下次分配簇数目>] |
        [MINEXTENTS <最小保留簇数目>] |
        [ON <表空间名>] |
        [FILLFACTOR <填充比例>]|
        [BRANCH <BRANCH 数>]|
        [BRANCH (<BRANCH 数>, <NOBRANCH 数>)]|
        [NOBRANCH]|
        [<CLUSTERBTR>]|
        [WITH COUNTER]|
        [WITHOUT COUNTER] |
```

```
        [USING LONG ROW]
    <存储加密子句> ::= <存储加密子句 1>|<存储加密子句 2>
    <存储加密子句 1> ::= ENCRYPT [<加密用法>|<加密用法><加密模式>|<加密模式>]
    <存储加密子句 2> ::= ENCRYPT { <加密用法>|<加密用法><加密模式>|<加密模式>}<散列选项>
    <加密用法> ::= WITH <加密算法>
    <加密模式> ::= <透明加密模式> | <半透明加密模式>
    <透明加密模式> ::= AUTO
    <半透明加密模式> ::= MANUAL
    <散列选项> ::= HASH WITH [<密码引擎名>.]<散列算法> [<加盐选项>]
    <加盐选项> ::= [NO] SALT
    <加密算法> ::= <DES_ECB | DES_CBC | DES_CFB|DES_OFB|DESEDE_ECB|
                DESEDE_CBC | DESEDE_CFB|DESEDE_OFB | AES128_ECB |
                AES128_CBC | AES128_CFB | AES128_OFB | AES192_ECB |
                AES192_CBC | AES192_CFB | AES192_OFB | AES256_ECB |
                AES256_CBC | AES256_CFB | AES256_OFB | RC4>
    <散列算法> ::= <MD5 | SHA1 | SHA224 | SHA256 | SHA384 | SHA512>

    <表级约束定义>::=[CONSTRAINT <约束名>] <表级约束子句>[<失效生效选项>]
    <表级约束子句>::=<表级完整性约束>
    <表级完整性约束> ::=
        <唯一性约束选项> (<列名> {,<列名>}) [USING INDEX TABLESPACE{ <表空间名> |
DEFAULT}]|
        FOREIGN KEY (<列名>{,<列名>}) <引用约束> |
        CHECK (<检验条件>)
    <空间限制子句> ::=
        DISKSPACE LIMIT <空间大小>|
        DISKSPACE UNLIMITED
    <压缩子句> ::=
        COMPRESS |
        COMPRESS (<列名> {,<列名>}) |
        COMPRESS EXCEPT (<列名> {,<列名>})
    <高级日志子句> ::= WITH ADVANCED LOG
    <add_log 子句>::= ADD LOGIC LOG
    <DISTRIBUTE 子句> ::=
        DISTRIBUTED[<RANDOMLY>|<FULLY>]|
        DISTRIBUTED BY [<HASH>](<列名> {,<列名>})|
        DISTRIBUTED BY RANGE (<列名> {,<列名>})(<范围分布项> {,<范围分布项>})|
        DISTRIBUTED BY LIST (<<列名> {,<列名>}>)(<列表分布项> {,<列表分布项>})
    <范围分布项> ::=
        VALUES LESS THAN (<表达式>{,<表达式>}) ON <实例名>|
        VALUES EQU OR LESS THAN (<表达式>{,<表达式>}) ON <实例名>
    <列表分布项> ::= VALUES (<表达式>{,<表达式>}) ON <实例名>
    <不带 INTO 的 SELECT 语句> ::= <查询表达式>|<带参数查询语句>
    <带参数查询语句>::=<子查询>|(<带参数查询语句>)
```

参数

1. <模式名> 指明该表属于哪个模式，缺省为当前模式；

2. <表名> 指明被创建的基表名，基表名最大长度 128 字节；

3. <列名> 指明基表中的列名，列名最大长度 128 字节；

4. <数据类型> 指明列的数据类型；

5. <列缺省值表达式> 如果之后的 INSERT 语句省略了插入的列值，那么此项为列值指定一个缺省值，可以通过 DEFAULT 指定一个值。DEFAULT 表达式串的长度不能超过 2048 字节；

6. <列级完整性约束定义>中的参数：

   1) NULL 指明指定列可以包含空值，为缺省选项。
   2) NOT NULL 非空约束，指明指定列不可以包含空值；
   3) UNIQUE 唯一性约束，指明指定列作为唯一关键字；
   4) PRIMARY KEY 主键约束，指明指定列作为基表的主关键字；
   5) CLUSTER PRIMARY KEY 主键约束，指明指定列作为基表的聚集索引（也叫聚簇索引）主关键字；
   6) NOT CLUSTER PRIMARY KEY 主键约束，指明指定列作为基表的非聚集索引主关键字；
   7) CLUSTER KEY 指定列为聚集索引键，但是是非唯一的；
   8) CLUSTER UNIQUE KEY 指定列为聚集索引键，并且是唯一的；
   9) USING INDEX TABLESPACE <表空间名> 指定索引存储的表空间；
   10) REFERENCES 指明指定列的引用约束。引用约束要求引用对应列类型必须基本一致。所谓基本，是因为 CHAR 与 VARCHAR，BINARY 与 VARBINARY，TINYINT、SMALLINT 与 INT 在此被认为是一致的。如果有 WITH INDEX 选项，则为引用约束建立索引，否则不建立索引，通过其他内部机制保证约束正确性；
   11) CHECK 检查约束，指明指定列必须满足的条件；
   12) NOT VISIBLE 列不可见，当指定某列不可见时，使用 SELECT *进行查询时将不添加该列作为选择列。不可见列不能创建索引；使用 INSERT 无显式指定列列表进行插入时，值列表不能包含隐藏列的值。

7. <表级完整性约束>中的参数：

   1) UNIQUE 唯一性约束，指明指定列或列的组合作为唯一关键字；
   2) PRIMARY KEY 主键约束，指明指定列或列的组合作为基表的主关键字。指明 CLUSTER，表明是主关键字上聚集索引；指明 NOT CLUSTER，表明是主关键字上非聚集索引；
   3) USING INDEX TABLESPACE <表空间名> 指定索引存储的表空间；
   4) FOREIGN KEY 指明表级的引用约束，如果使用 WITH INDEX 选项，则为引用约束建立索引，否则不建立索引，通过其他内部机制保证约束正确性；
   5) CHECK 检查约束，指明基表中的每一行必须满足的条件；
   6) 与列级约束之间不应该存在冲突。

8. ON COMMIT<DELETE | PRESERVE>ROWS 用来指定临时表（TEMPORARY）中的数据是事务级或会话级的，缺省情况下是事务级。ON COMMIT DELETE ROWS：指定临时表是事务级的，每次事务提交或回滚之后，表中所有数据都被删除；ON COMMIT PRESERVE ROWS：指定临时表是会话级的，会话结束时才清空表；

9. <检验条件> 指明表中一列或多列能否接受的数据值或格式；

10. <查询表达式>和<子查询> 定义请查看数据查询章节；

11. STORAGE 项中：

BRANCH、NOBRANCH 是堆表创建关键字，堆表为“扁平 B 树表”。这两个参数用来指定堆表并发分支 BRANCH 和非并发分支 NOBRANCH 的数目。<BRANCH 数>取值范围为 1~64，<NOBRANCH 数>取值范围为 1~64。

1) NOBRANCH：指定创建的表为堆表，并发分支个数为 0，非并发分支个数为 1；
2) BRANCH (<BRANCH 数>, <NOBRANCH 数>)：指定创建的表为堆表，并发分支个数为<BRANCH 数>，非并发个数为<NOBRANCH 数>；
3) BRANCH <BRANCH 数>：指定创建的表为堆表，并发分支个数为<BRANCH 数>，非并发分支个数为 0。

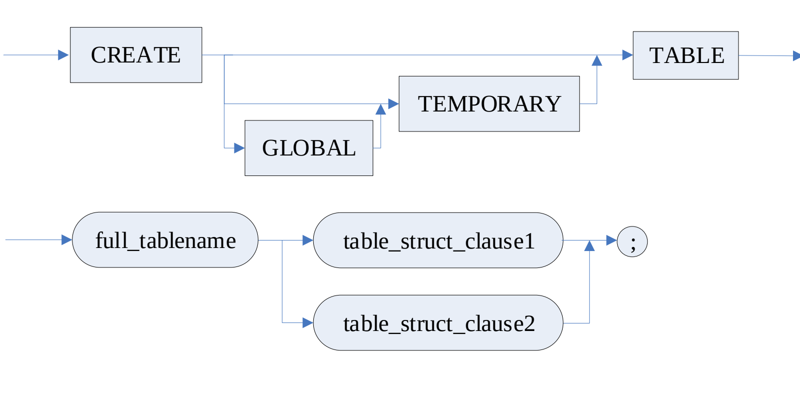
12. CLUSTERBTR 指定创建的表为非堆表，即普通 B 树表；

13. <虚拟列定义>指明定义虚拟列的表达式；

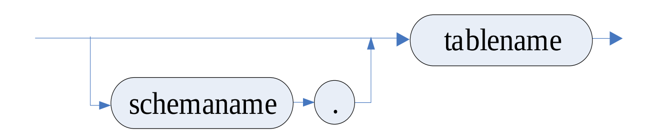
14. <高级日志子句>指定创建日志辅助表，具体可参考 19.3.1.1；

15. <add_log 子句> 开启表的物理逻辑日志记录功能，缺省为不开启。<add_log 子句>开启时和 INI 参数 RLOG_IGNORE_TABLE_SET=1 时功能一样。

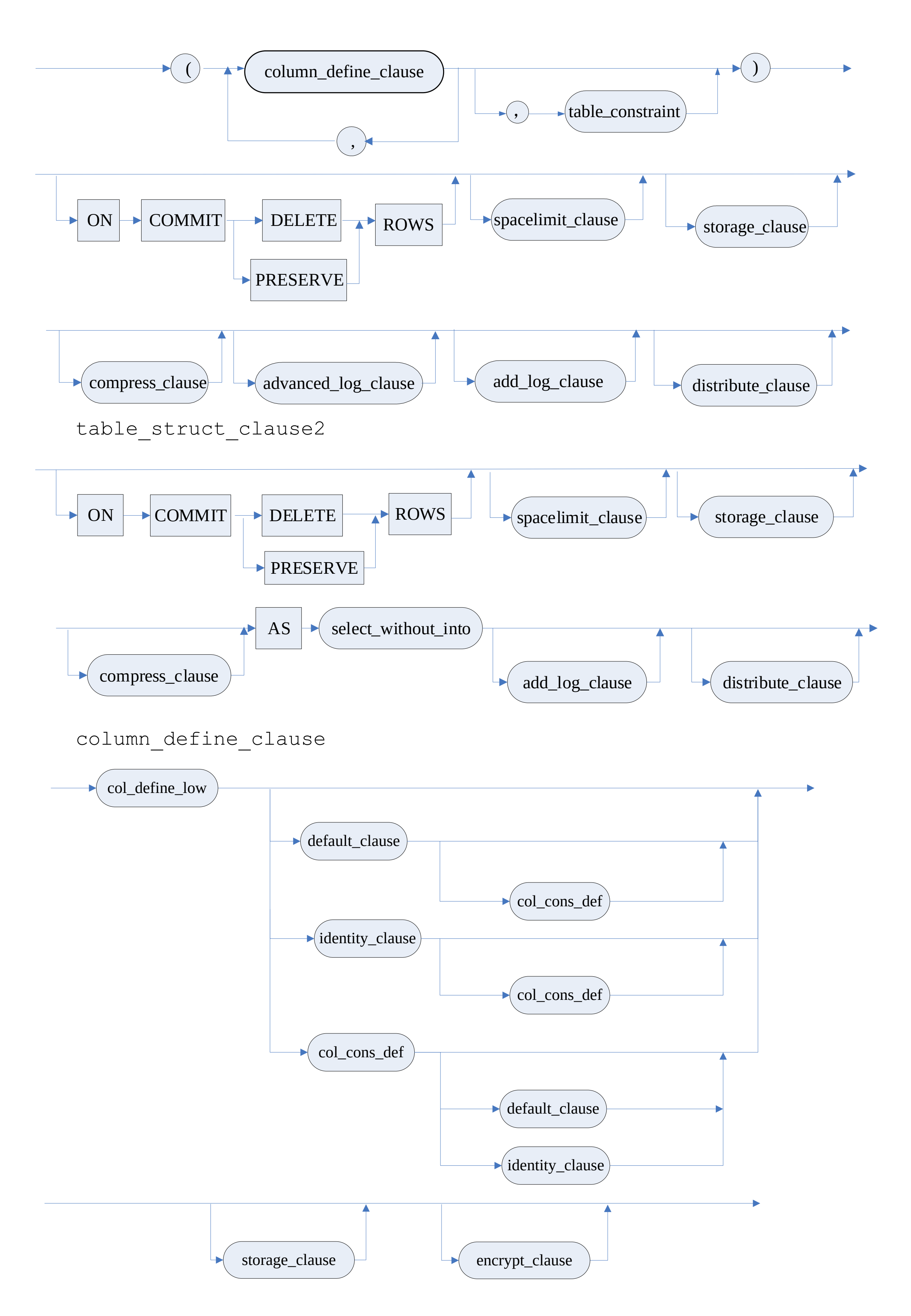
图例



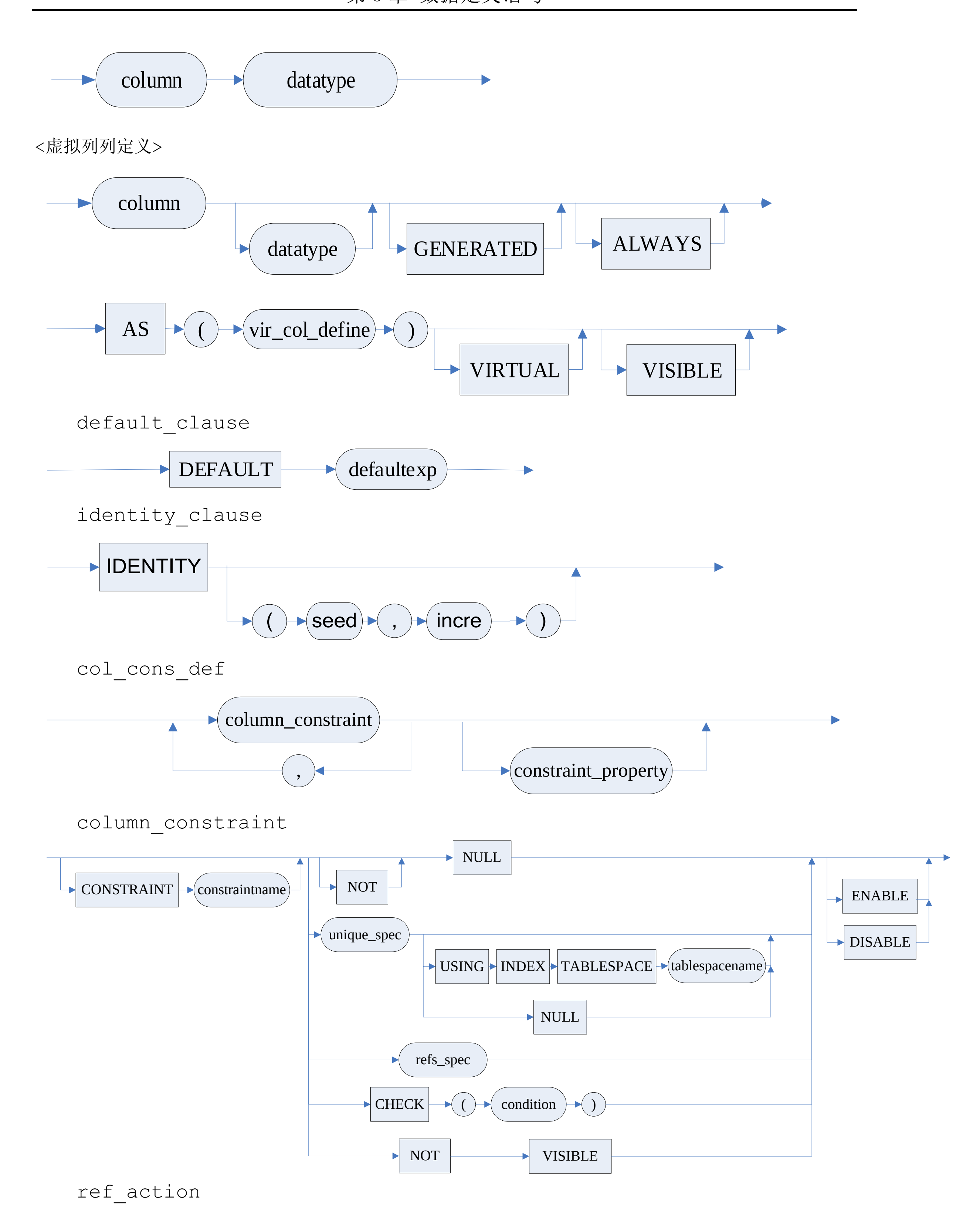
full_tablename



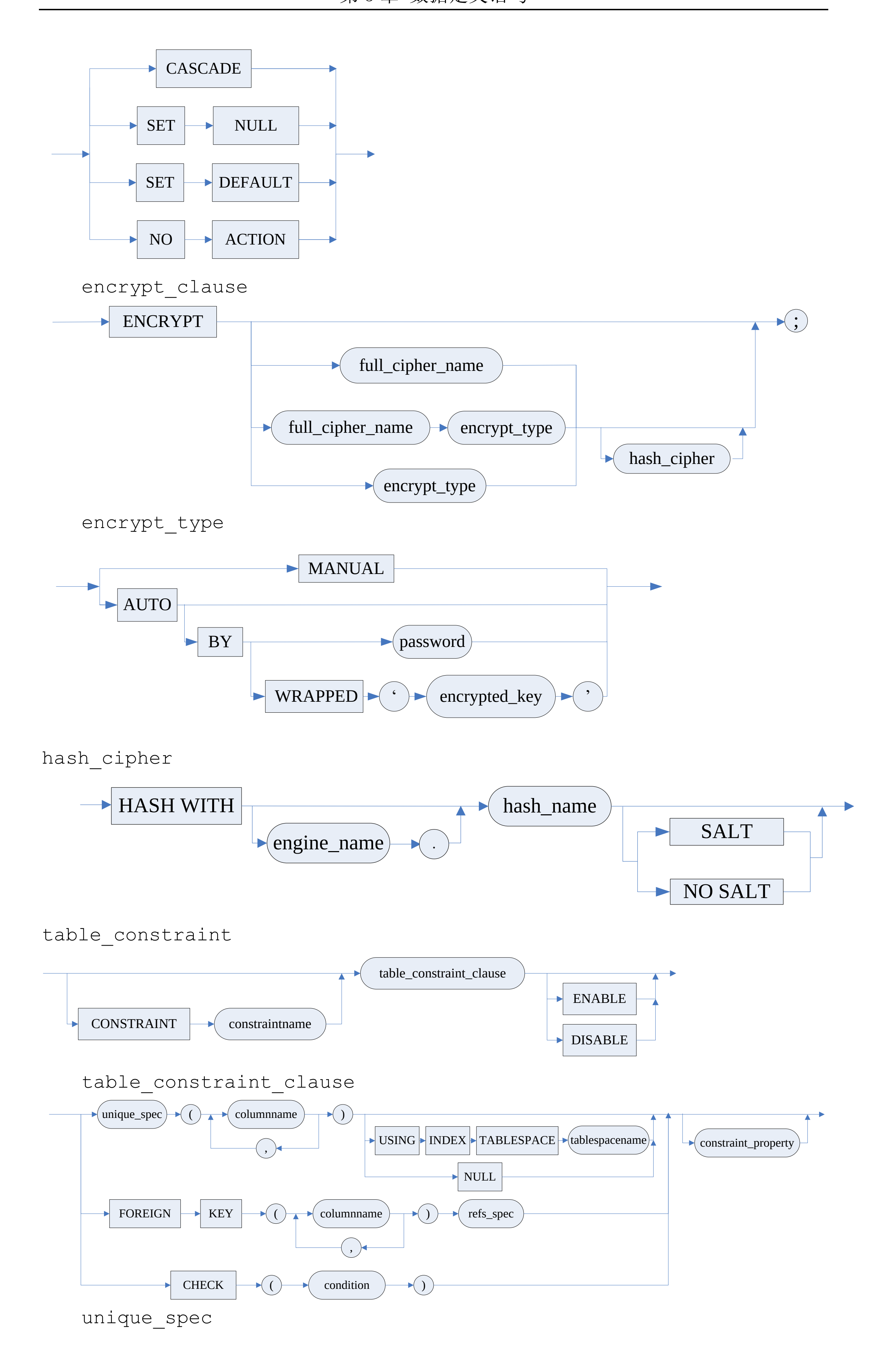
table_struct_clause1

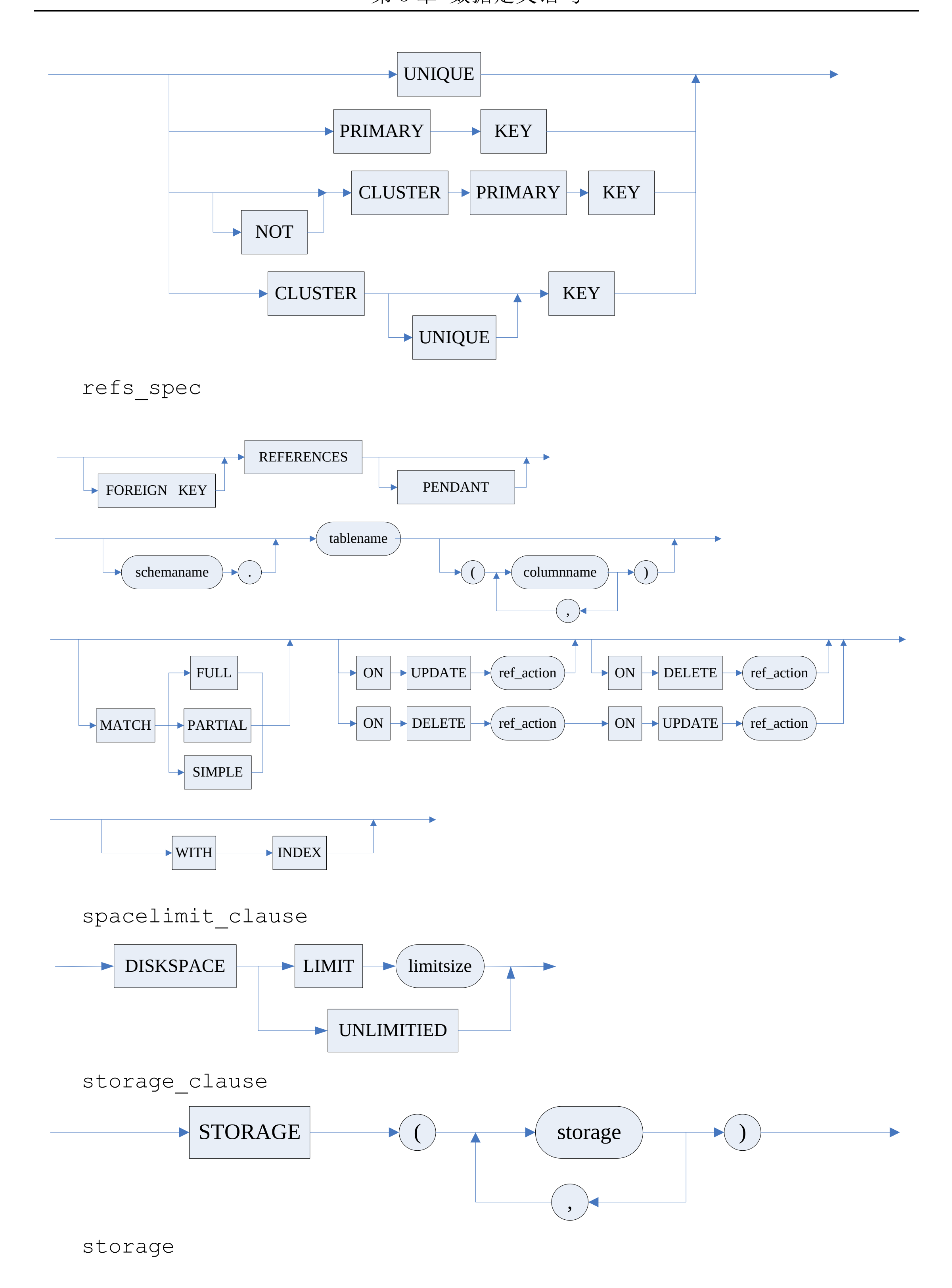

(
column_define_clause
,
,
table_constraint
)
ON
COMMIT
DELETE
PRESERVE
ROWS
spacelimit_clause
storage_clause
compress_clause
advanced_log_clause
add_log_clause
distribute_clause
table_struct_clause2
ON
COMMIT
DELETE
PRESERVE
ROWS
spacelimit_clause
storage_clause
compress_clause
AS
select_without_into
add_log_clause
distribute_clause
column_define_clause
col_define_low
default_clause
col_cons_def
identity_clause
col_cons_def
col_cons_def
default_clause
identity_clause
storage_clause
encrypt_clause

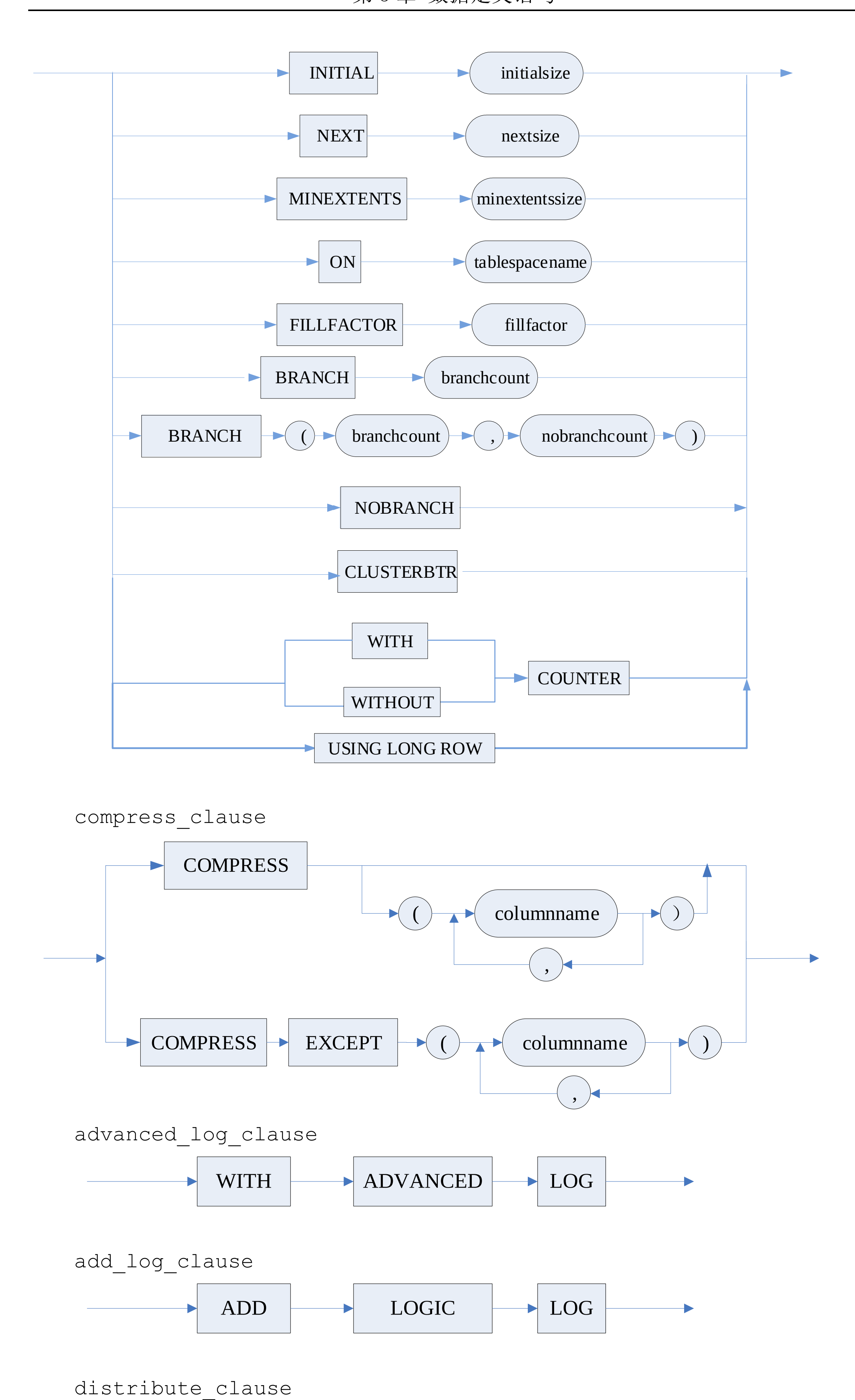
<普通列定义>

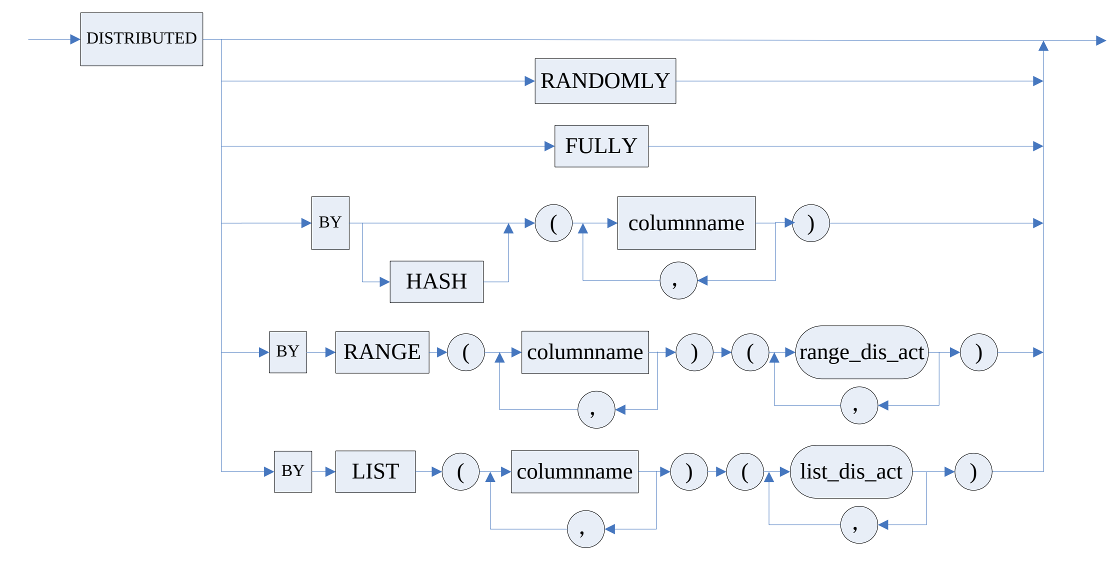


column
datatype
<虚拟列列定义>
column
datatype
GENERATED
ALWAYS
AS
(
vir_col_define
)
VIRTUAL
VISIBLE
default_clause
DEFAULT
defaultexp
identity_clause
IDENTITY
(
seed
,
incre
)
col_cons_def
column_constraint
,
constraint_property
column_constraint
CONSTRAINT
constraintname
NULL
NOT
unique_spec
USING
INDEX
TABLESPACE
tablespacename
NULL
refs_spec
CHECK
(
condition
)
NOT
VISIBLE
ENABLE
DISABLE
ref_action

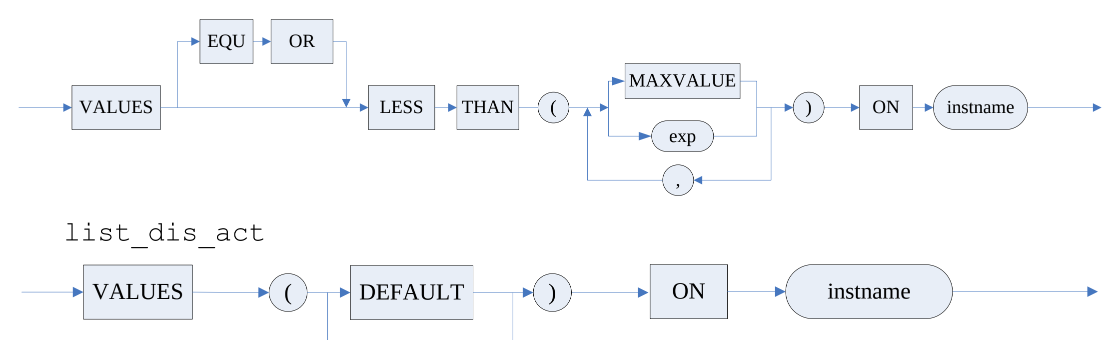

CASCADE
SET
NULL
SET
DEFAULT
NO
ACTION
encrypt_clause
ENCRYPT
;
full_cipher_name
full_cipher_name
encrypt_type
hash_cipher
encrypt_type
encrypt_type
MANUAL
AUTO
BY
password
WRAPPED
‘
encrypted_key
’
hash_cipher
HASH WITH
hash_name
engine_name
.
SALT
NO SALT
table_constraint
table_constraint_clause
CONSTRAINT
constraintname
ENABLE
DISABLE
table_constraint_clause
unique_spec
(
columnname
)
,
USING
INDEX
TABLESPACE
tablespacename
NULL
constraint_property
FOREIGN
KEY
(
columnname
)
refs_spec
,
CHECK
(
condition
)
unique_spec


UNIQUE
PRIMARY
KEY
NOT
CLUSTER
PRIMARY
KEY
CLUSTER
KEY
UNIQUE
refs_spec
FOREIGN KEY
REFERENCES
PENDANT
schemaname
.
tablename
(
columnname
)
,
MATCH
FULL
PARTIAL
SIMPLE
ON
UPDATE
ref_action
ON
DELETE
ref_action
ON
DELETE
ref_action
ON
UPDATE
ref_action
WITH
INDEX
spacelimit_clause
DISKSPACE
LIMIT
limitsize
UNLIMITIED
storage_clause
STORAGE
(
storage
,
)
storage


INITIAL
initialsize
NEXT
nextsize
MINEXTENTS
minextentssize
ON
tablespacename
FILLFACTOR
fillfactor
BRANCH
branchcount
BRANCH
(
branchcount
,
nobranchcount
)
NOBRANCH
CLUSTERBTR
WITH
WITHOUT
COUNTER
USING LONG ROW
compress_clause
COMPRESS
(
columnname
,
)
COMPRESS
EXCEPT
(
columnname
,
)
advanced_log_clause
WITH
ADVANCED
LOG
add_log_clause
ADD
LOGIC
LOG
distribute_clause



range_dis_act



list_dis_act

VALUES (DEFAULT exp ,) ON instname

语句功能

供 DBA 或具有 CREATE_TABLE 权限的用户定义基表。

使用说明

1． <表名>指定了所要建立的基表名。在一个<模式>中，<基表名>、<视图名>均不相同。如果<模式名>缺省，则缺省为当前模式。若指定 TEMPORARY，则表示该表为一个临时表，只在一个会话中有效，当一个会话结束，该临时表被自动清空。表名需要是合法的标识符，且满足 SQL 语法要求。当表名以"##"开头时，该表为全局临时表；

2． TEMPORARY 临时表不支持压缩 compress 功能；

3． GLOBAL 目前仅支持 GLOBAL 临时表，因此建临时表时是否指定 GLOBAL 效果是一样的；

4． 表名最大长度为 128 个字节；

5． 所建基表至少要包含一个<列名>指定的列，在一个基表中，各<列名>不得相同。一张基表中至多可以包含 2048 列；

6． 虚拟列上存在约束时，不支持使用 dmfldr 导入；

7． <DEFAULT 子句>指定列的缺省值，如：DEFAULT DATE '2015-12-26'；

8． 如果未指明 NOT NULL，也未指明<DEFAULT 子句>，则隐含为 DEFAULT NULL；

9． 自增列不能使用<DEFAULT 子句>；

10．<列缺省值表达式>的数据类型必须与本列的<数据类型>一致。缺省值表达式存在

以下几点约束：

1） 仅支持只读系统函数或指定 FOR CALCULATE 创建的存储函数；
2） 不支持表列；
3） 不支持包变量或语句参数；
4） 不支持查询表达式；
5） 不支持 LIKE；
6） 不支持 CONTAINS 表达式。

11. 如果列定义为 NOT NULL，则当该列插入空值时会报错；

12. 约束被 DM 用来对数据实施业务规则，完成对数据完整性的控制。DM_SQL 中主要定义了以下几种类型的约束：非空约束、唯一性约束、主键约束、引用约束和检查约束。如果完整性约束只涉及当前正在定义的列，则既可定义成列级完整性约束，也可以定义成表级完整性约束；如果完整性约束涉及到该基表的多个列，则只能在语句的后面定义成表级完整性约束。

定义与该表有关的列级或表级完整性约束时，可以用 CONSTRAINT<约束名>子句对约束命名，系统中相同模式下的约束名不得重复。如果不指定约束名，系统将为此约束自动命名。经定义后的完整性约束被存入系统的数据字典中，用户操作数据库时，由 DBMS 自动检查该操作是否违背这些完整性约束条件。

1） 非空约束主要用于防止向一列添加空值，这就确保了该列在表中的每一行都存在一个有意义的值。
   a） 该约束仅用于列级；
   b） 如果定义了列约束为 NOT NULL，则其<列缺省值表达式>不能将该列指定为 NULL；
   c） 空值即为未知的值，没有大小，不可比较。除关键字列外，其列可以取空值。不可取空值的列要用 NOT NULL 进行说明。
2） 唯一性约束主要用于防止一个值或一组值在表中的特定列里出现不止一次，确保数据的完整性。
   a） 唯一性约束是通过唯一索引来实现的。创建了一个唯一索引也就创建了一个唯一性约束；同样的，创建了一个唯一性约束，也就同时创建了一个唯一索引，这种情况下唯一索引是由系统自动创建的；
   b） NULL 值是不参加唯一性约束的检查的。DM 系统允许插入多个 NULL 值。对于组合的唯一性约束，只要插入的数据中涉及到唯一性约束的列有一个或多个 NULL 值，系统则认为这笔数据不违反唯一性约束。
3） 主键约束确保了表中构成主键的一列或一组列的所有值是唯一的。主键主要用于识别表中的特定行。主键约束是唯一性约束的特例。
   a） 可以指定多个列共同组成主键，最多支持 63 个列；
   b） 主键约束涉及的列必须为非空。通常情况下，DM 系统会自动在主键约束涉及的列上自动创建非空约束；
   c） 每个表中只能有一个主键；
   d） 主键约束是通过创建唯一索引来实现的。DM 系统允许用户自己定义创建主键时，通过 CLUSTER 或 NOT CLUSTER 关键字来指明创建索引的类型。CLUSTER 指明该主键是创建在聚集索引上的，NOT CLUSTER 指明该主键是创建在非聚集索引上的。缺省情况下，主键是创建在非聚集索引上的。堆表和列存储表不允许建立聚集主键；
   e） 表中不能同时包含聚集主键和多媒体数据类型列。

4) 引用约束用于保证相关数据的完整性。引用约束中构成外键的一列或一组列，其值必须至少匹配其参照的表中的一行的一个主键或唯一键值。我们把这种数据的相关性称为引用关系，外键所在的表称为引用表，外键参照的表称为被引用表。
   a) 引用约束指明的被引用表上必须已经建立了相关主键或唯一索引。也就是说，必须保持引用约束所引用的数据必须是唯一的；
   b) 引用约束的检查规则：
      i. 插入规则：外键的插入值必须匹配其被引用表的某个键值。
      ii. 更新规则：外键的更新值必须匹配被引用表的某个键值。当修改被引用表中的主键值时，如果定义约束时的选项是 NO ACTION，且更新结果会违反引用约束则不允许更新；如果定义的是 SET NULL 则将引用表上的相关外键值置为 NULL；如果定义的是 CASCADE，那么引用表上的相关外键值将被修改为同样的值；如里定义的是 SET DEFAULT，则把引用列置为该列的缺省值。
      iii. 删除规则：当从被引用表中删除一行数据时，如果定义约束时的选项是 NO ACTION，就不删除引用表上的相关外键值；如果定义的是 SET NULL 则将引用表上的相关外键值置为 NULL；如果定义的是 CASCADE，那么引用表上的相关外键值将被删除；如里定义的是 SET DEFAULT，则把每个引用列置为"<列缺省值表达式>"规则中所指定的缺省值。
   c) NULL 值不参加引用约束的检查。受引用约束的表，如果要插入的涉及到引用约束的列值有一个或多个 NULL，则认为插入值不违反引用约束；
   d) MPP 环境下，引用列和被引用列都必需包含分布列，且分布情况完全相同；
   e) MPP 环境下，不支持创建 SET NULL 或 SET DEFAULT 约束检查规则的引用约束。
5) 检查约束用于对将要插入的数据实施指定的检查，从而保证表中的数据都符合指定的限制。<检验条件>必须是一个有意义的布尔表达式，其中的每个列名必须是本表中定义的列，但列的类型不得为多媒体数据类型，并且不应包含子查询、集函数。

13. 可以使用空间限制子句 DISKSPACE LIMIT 来限制表的最大存储空间，以 M 为单位，取值范围为 1 到 1048576，关键字 UNLIMITED 表示无限制。系统不支持查询建表情况下指定空间限制；

14. 可以使用 STORAGE 子句指定表的存储信息：
1) 初始簇数目：指建立表时分配的簇个数，必须为整数，最小值为 1，最大值为 256，缺省为 1；
2) 下次分配簇数目：指当表空间不够时，从数据文件中分配的簇个数，必须为整数，最小值为 1，最大值为 256，缺省为 1；
3) 最小保留簇数目：当删除表中的记录后，如果表使用的簇数目小于这个值，就不再释放表的空间，必须为整数，最小值为 1，最大值为 256，缺省为 1；
4) 表空间名：在指定的表空间上建表，表空间必须已存在，缺省为该用户的默认表空间；
5) 填充比例：指定存储数据时每个数据页和索引页的充满程度，取值范围从 0 到 100。默认值为 0，等价于 100，表示全满填充。插入数据时填充比例的值越低，可由新数据使用的空间就越多；更新数据时填充比例的值越大，更新导致

出现的页分裂的几率越大。同样，创建索引时，填充比例的值越低，可由新索引项使用的空间也就越多；

6) BRANCH 和 NOBRANCH：指定 BRANCH 和 NOBRANCH 的个数；
7) CLUSTERBTR：当 INI 参数 LIST_TABLE = 1 时，指定 CLUSTERBTR，则建立的表为普通 B 树表而非堆表；
8) WITH COUNTER：在表上维护当前表内的行数；WITHOUT COUNTER：表上只维护一个非实时的大概的行数；
   对用户的影响：例如 SELECT COUNT(*) FROM test；如果表 test 是 WITH COUNTER 属性，服务器直接取行数返回即可，可以快速响应；如果表 test 是 WITHOUT COUNTER 属性，服务器需要先扫描 B 树获取行数返回后才能响应。不同的场景，根据需要灵活选择 COUNTER 属性。WITH COUNTER 属性可以通过 ALTER TABLE 语句修改。若省略该选项，默认是 WITH COUNTER 属性。
9) USING LONG ROW：支持超长记录存储。当 DM 行存储的记录长度超过页大小一半时，先尝试将过长的变长字符串转换为行外 BLOB 存储，如果转换后仍超长则报错。临时表、HUGE 表、外部表不支持 USING LONG ROW 选项。水平分区子表的 USING LONG ROW 选项自动采用与主表保持一致的方式，两者不同的情况下，直接忽略水平分区子表的 USING LONG ROW 选项。

15. <压缩子句> 只是语法支持，功能已经取消；

16. 记录的列长度总和不超过块长的一半，VARCHAR 数据类型的长度是指数据定义长度，实际是否越界还需要判断实际记录的长度，而 CHAR 类型的长度是实际数据长度。与此类似的还有 VARBINARY 和 BINARY 数据类型。因此，对于 16K 的块，可以定义 CREATE TABLE TEST(C1 VARCHAR(8000),C2 INT)，但是不能定义 CREATE TABLE TEST(C1 CHAR(8000)，C2 INT)；

17. DM 具备自动断句功能；

18. 在对列指定存储加密属性时，对用户的数据在保存到物理介质之前使用指定的加密算法加密，防止数据泄露；

19. 加密算法可以是系统中已经存在的算法名称，可选的算法可以在 V$CIPHERS 中获取，也可以使用第三方加密库中的算法，第三方加密库的实现可参考《DM8 安全管理》的相关章节，将已实现的第三方加密动态库放到 bin 目录下的文件夹 external_crypto_libs 中，DM 支持加载多个第三方加密动态库，然后重启 DM 服务器即可引用其中的算法；

需要注意的是：以“NOPAD”结尾的加密算法需要用户保证原始数据长度是 BLOCK_SIZE 的整数倍，DM 不会自动填充。如果数据不一定是 BLOCK_SIZE 的整数倍，请选择不以“NOPAD”结尾的加密算法，以“NOPAD”结尾的加密算法主要用于数据页分片加密。

20. 散列算法。散列算法用于保证用户数据的完整性，若用户数据被非法修改，则能判断该数据不合法。加盐选项可以与散列算法配合使用；

21. 透明加密模式。用透明加密的方式加密列上的数据，在数据库中保存加密该列的密钥，执行 DML 语句的过程中自动获取密钥；

22. 可以使用 DISTRIBUTE 子句指定表的分布类型：

1) 单机模式下建的分布表和普通表一样，但是不能创建指定实例名的分布表（如范围分布表和 LIST 分布表）。
2) 在 MPP 模式下建分布表，如果未指定列则默认为 RANDOMLY(随机)分布表。

3) 分布列类型不支持 BLOB、CLOB、IMAGE、TEXT、LONGVARCHAR、BIT、BINARY、VARBINARY、LONGVARBINARY、时间间隔类型和用户自定义类型。
4) HASH 分布、RANGE 分布、LIST 分布允许更新分布列，并支持包含大字段列的表的分布列更新，但包含 INSTEAD OF 触发器的表、堆表不允许更新分布列。
5) 对于 FULLY（复制）分布表，只支持单表查询的更新和删除操作，并且查询项或者条件表达式中都不能包含 ROWID 伪列表达式。
6) RANGE（范围）分布表和 LIST（列表）分布表，分布列与分布列值列表必须一致，并且指定的实例名不能重复。
7) 随机分布表不支持 UNIQUE 索引。

23. 虚拟列的使用：

虚拟列的值是不存储在磁盘上的，而是在查询的时候，根据定义的表达式临时计算后得到的结果。虚拟列可以用在查询、DML、DDL 语句中。索引可以建在虚拟列上。用户可以像使用普通列一样使用虚拟列。

1) GENERATED ALWAYS 和 VIRTUAL 为可选关键字，主要用于描述虚拟列的特性，写与不写没有本质区别。
2) 虚拟列中的 VISIBLE 只是语法支持，没有实际意义。
3) 不支持在索引表、外部表、临时表上使用虚拟列。虚拟列和虚拟列中使用的列必须来自同一个表。表中至少要有一个非虚拟列。
4) 在虚拟列上建索引相当于在表上建函数索引。
5) 即使表达式中的列已经有列级安全属性，虚拟列也不会继承列的安全规则。因此，为了保护虚拟列的数据，可以复制列级安全策略或者使用函数隐式来保护虚拟列数据。例如，信用卡的卡号被列级安全策略保护，只允许员工看到卡号的后四位。在这种情况下，可以把信用卡号的后四位定义成虚拟列。
6) 虚拟列不能作为分布表的分布列。
7) 虚拟列之间不能嵌套定义。
8) 虚拟列不能作为引用列。
9) 虚拟列最后的输出应该是标量性的。
10) 虚拟列不能是用户自定义类型、大字段类型。
11) 不能直接插入、更新虚拟列；但是可以在更新和删除的 WHERE 子句中使用虚拟列。
12) 不能删除被虚拟列引用的实际列。表中只有一个实际列时，这个实际列不能被删除。
13) 虚拟列不能被修改为实际列。实际列也不能被修改为虚拟列。
14) 虚拟列不能设置 DEFAULT 值。
15) 虚拟列不能设置为 IDENTITY。
16) 不支持在虚拟列上建立位图连接索引和全文索引。
17) 不支持在虚拟列上建立物化视图日志。
18) 虚拟列不能在创建时作为 CLUSTER PK、UNIQUE 或者 PK。但是可以在虚拟列上创建 UNIQUE INDEX。
19) 虚拟列不支持 PLUS JOIN。
20) 虚拟列不支持统计信息。
21) 虚拟列不支持加密。
22) 虚拟列的表达式定义长度不能超过 2048 字节。

23）DM 支持系统内部函数，但是不能保证数据的确定性 。
24）虚拟列表达式中不支持 ROW LIKE 表达式。
25）虚拟列表达式不支持 CONTAINS 表达式。
26）如果虚拟列表达式包含函数，则必须是确定性的函数。不允许使用分组函数、子查询（集函数、分析函数都不允许）。
27）更改表增加多列时，虚拟列表达式不能使用新增加的列。
28）虚拟列上有约束时，使用 dmfldr 导入，将不能保证正确性。
29）虚拟列上创建有索引时，则该虚拟列不支持修改列数据类型和列表达式。

举例说明

例 1 首先回顾一下第二章中定义的基表，它们均是用列级完整性约束定义的格式写出，也可以将唯一性约束、引用约束和检查约束以表级完整性约束定义的格式写出的。假定用户为 SYSDBA，下面以产品的评论表为例进行说明。

```
CREATE  TABLE PRODUCTION.PRODUCT_REVIEW
(
 PRODUCT_REVIEWID INT IDENTITY(1,1),
 PRODUCTID INT NOT NULL,
 NAME VARCHAR(50) NOT NULL,
 REVIEWDATE DATE NOT NULL,
 EMAIL VARCHAR(50) NOT NULL,
 RATING INT NOT NULL,
 COMMENTS TEXT,
 PRIMARY KEY(PRODUCT_REVIEWID),
 FOREIGN  KEY(PRODUCTID)  REFERENCES  PRODUCTION.PRODUCT(PRODUCTID),
 CHECK(RATING IN(1,2,3,4,5))
);
--注：该语句的执行需在"产品的信息表"已经建立的前提下
```

系统执行建表语句后，就在数据库中建立了相应的基表，并将有关基表的定义及完整性约束条件存入数据字典中。需要说明的是，由于被引用表要在引用表之前定义，本例中的产品的信息表被产品的评论表引用，所以这里应先定义产品的信息表，再定义产品的评论表，否则就会出错。

例 2 建表时指定存储信息，表 PERSON 建立在表空间 FG_PERSON 中，初始簇大小为 5，最小保留簇数目为 5，下次分配簇数目为 2，填充比例为 85。

```
CREATE TABLESPACE FG_PERSON DATAFILE 'FG_PERSON.DBF' SIZE 128;

CREATE TABLE PERSON.PERSON
( PERSONID  INT IDENTITY(1,1) CLUSTER PRIMARY KEY,
SEX CHAR(1) NOT NULL,
NAME VARCHAR(50) NOT NULL,
EMAIL VARCHAR(50),
PHONE VARCHAR(25))
STORAGE
( INITIAL     5,
MINEXTENTS   5,
NEXT    2,
```

```
ON      FG_PERSON,
FILLFACTOR   85);
```

例 3 建立如下范围分布表后，表 PRODUCT_INVENTORY 将按照 QUANTITY 列值，被分布到 2 个站点上。

```
CREATE TABLE PRODUCTION.PRODUCT_INVENTORY
(PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
LOCATIONID INT NOT NULL REFERENCES PRODUCTION.LOCATION(LOCATIONID),
QUANTITY INT NOT NULL)
DISTRIBUTED BY RANGE (QUANTITY)
(
    VALUES EQU OR LESS THAN (100) ON EP01,
    VALUES EQU OR LESS THAN (MAXVALUE) ON EP02
);
```

例 4 建立如下列表分布表后，表 PRODUCT_INVENTORY 将按照 LOCATIONID 列值，被分布到 2 个站点上，1，2，3，4 在 EP01 上，5，6，7，8 在 EP02，如果有插入其它值时则报错。

```
CREATE TABLE PRODUCTION.PRODUCT_INVENTORY
(PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
LOCATIONID INT NOT NULL REFERENCES PRODUCTION.LOCATION(LOCATIONID),
QUANTITY INT NOT NULL)
DISTRIBUTED BY LIST (LOCATIONID)
(
    VALUES (1,2,3,4) ON EP01,
    VALUES (5,6,7,8) ON EP02
);
```

例 5 建立如下复制分布表后，表 LOCATION 被分布到 MPP 各个站点上，每个站点上的数据都保持一致。

```
CREATE TABLE PRODUCTION.LOCATION
(LOCATIONID INT IDENTITY(1,1) PRIMARY KEY,
PRODUCT_SUBCATEGORYID INT NOT NULL,
NAME VARCHAR(50) NOT NULL)
DISTRIBUTED  FULLY;
```

例 6 建立普通表查看 select count(*) 执行计划，再删除后重建表 test 带 without counter 属性，再查看执行计划：

```
SQL>create table test(c1 int);
操作已执行
已用时间:69.134(毫秒). 执行号:2422366.

SQL>explain select count(*) from test;


1   #NSET2: [0, 1, 0]
2     #PRJT2: [0, 1, 0]; exp_num(1), is_atom(FALSE)
3       #FAGR2: [0, 1, 0]; sfun_num(1)
```

```
已用时间:208.560(毫秒). 执行号:0.

SQL>drop table test;
操作已执行
已用时间:22.604(毫秒). 执行号:2446520.

SQL>create table test(c1 int) storage(without counter);
操作已执行
已用时间:198.693(毫秒). 执行号:2463231.

SQL>explain select count(*) from test;

1   #NSET2: [0, 1, 0]
2     #PRJT2: [0, 1, 0]; exp_num(1), is_atom(FALSE)
3       #AAGR2: [0, 1, 0]; grp_num(0), sfun_num(1)
4         #CSCN2: [0, 1, 0]; INDEX33555461(TEST)

已用时间:9.987(毫秒). 执行号:0.
```

3.6.1.2 定义外部表

需指定如下信息：

1. 表名、表所属的模式名；
2. 列定义；
3. 控制文件路径。

语法格式

```
CREATE EXTERNAL TABLE <表名定义> <表结构定义>;
<表名定义> ::=[<模式名>.]<表名>
<表结构定义> ::= (<列定义> {,<列定义>}) <FROM 子句>
<列定义> ::= <列名> <数据类型>
<列定义>  参见 3.6.1 节说明
<FROM 子句> = <FROM 子句 1> | <FROM 子句 2> | <FROM 子句 3>| <FROM 子句 4>
<FROM 子句 1> ::= FROM '<控制文件路径>'
<FROM 子句 2> ::= FROM DATAFILE '<数据文件路径>' [<数据文件参数列表>]
<FROM 子句 3> ::= FROM <控制文件目录> <控制文件选项>
<FROM 子句 4> ::= FROM DATAFILE <数据文件目录> <数据文件选项> [<数据文件参数列表>]
<数据文件参数列表> ::= PARMS(<参数选项> {,<参数选项>})
<参数选项> :: =
                [FIELDS DELIMITED BY <表达式>] |
                [RECORDS DELIMITED BY <表达式>]|
                [ERRORS <n>]|
                [BADFILE '<数据文件路径>']|
                [LOG <日志文件路径>] |
                [NULL_STR <NULL 字符串>] |
                [SKIP <跳过行数>] |
```

```
                [CHARACTER_CODE <文件字符集>]
<控制文件目录> ::= DEFAULT DIRECTORY <控制文件目录名>
<控制文件选项> ::= LOCATION ('<控制文件名>')
<数据文件目录> ::= DEFAULT DIRECTORY <数据文件目录名>
<数据文件选项> ::= LOCATION ('<数据文件名>')
```

参数

1．<模式名>　　指明该表属于哪个模式，缺省为当前模式；

2．<表名>　指明被创建的外部基表名；

3．<列名>　指明基表中的列名；

4．<数据类型>　指明列的数据类型，暂不支持多媒体类型；

5．<控制文件路径>　指明使用的控制文件的路径的字符串；

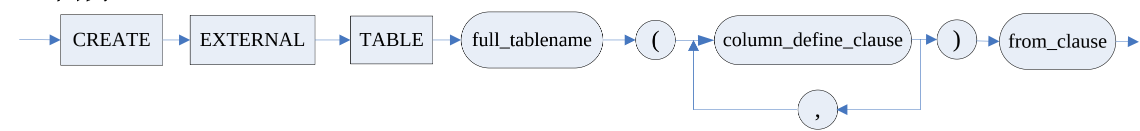
6．<数据文件路径>　指明使用的数据文件路径的字符串；

7．<参数选项>　FIELDS 表示列分隔符；RECORDS 表示行分隔符，缺省为回车；ERRORS 表示忽略外部表数据转换中出现错误的行数，取值范围为大于 0 的正整数，缺省为 0，表示不忽略错误。LOG 表示日志文件路径，如果不指定则不会生产日志；NULL_STR 指定数据文件中 NULL 值的表示字符串，默认忽略此参数；SKIP 指定跳过数据文件起始的逻辑行数，默认为 0；CHARACTER_CODE 指定数据文件中数据的编码格式，默认为 GBK，可选项有 GBK，UTF-8，SINGLE_BYTE 和 EUC-KR；

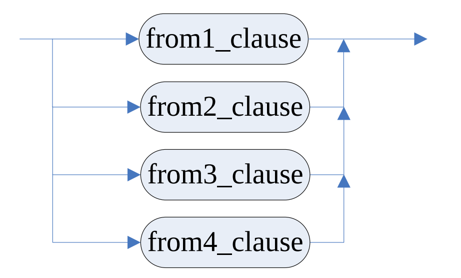
8．<表达式>　　字符串或十六进制串类型表达式；

9．<控制文件目录名>　　指数据库对象目录的名称。

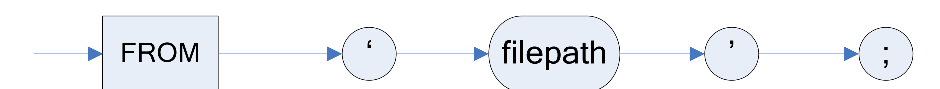




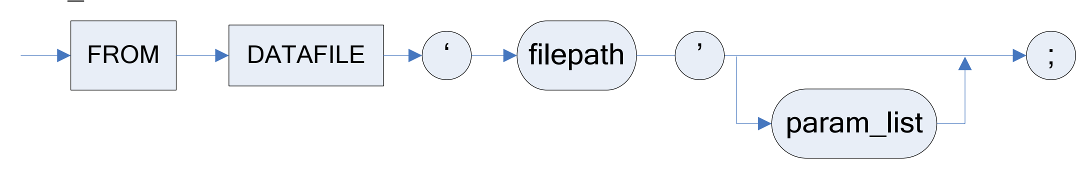
from_clause



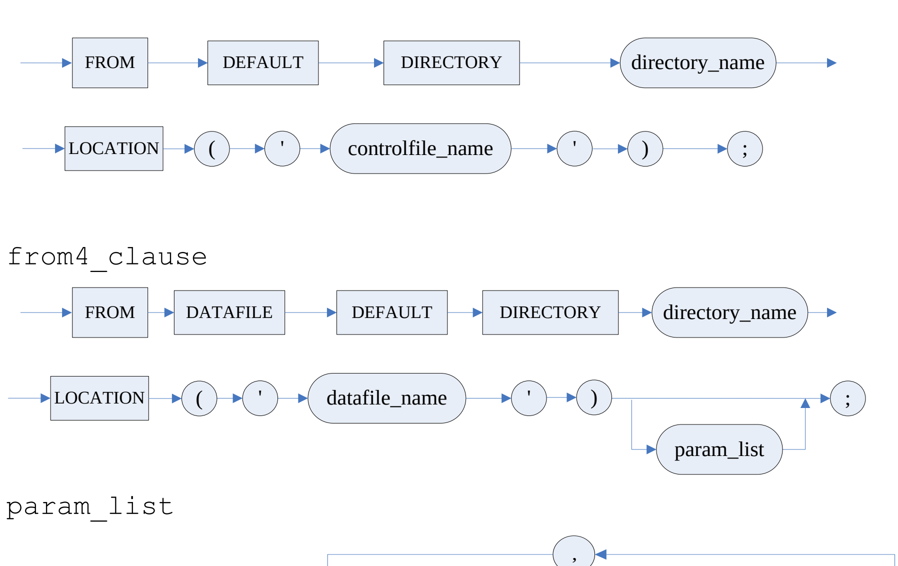
from1_clause



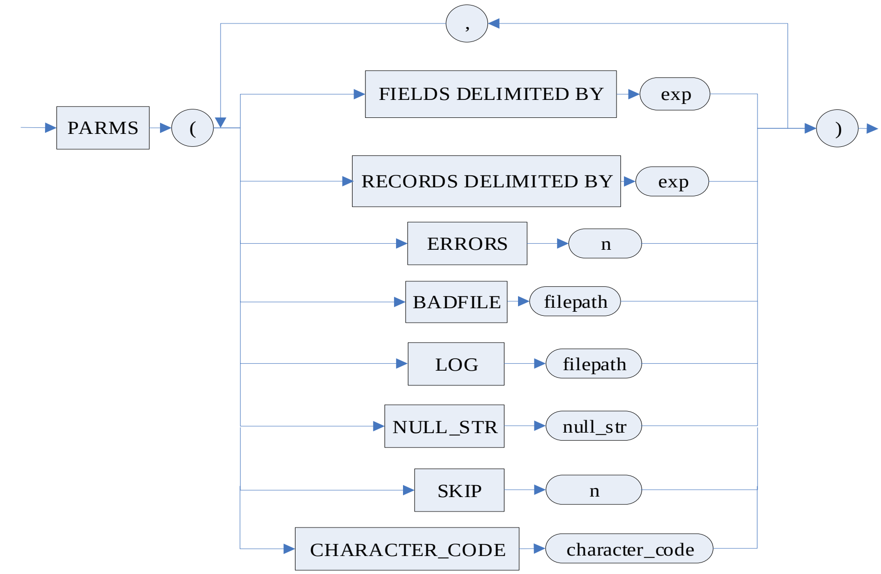
from2_clause



from3_clause



param_list



语句功能

供 DBA 或具有 CREATE_TABLE 权限的用户定义外部基表。MPP 环境下不支持创建外部表。

使用说明

1．<表名>指定了所要建立的外部基表名。如果<模式名>缺省，则缺省为当前模式。表名需要是合法的标识符，且满足 SQL 语法要求；

2．外部表的表名最大长度为 128 个字符；

3．所建外部基表至少要包含一个<列名>指定的列，在一个外部基表中，各<列名>不得相同。一张外部基表中至多可以包含 2048 列；

4．外部基表不能存在大字段列；

5．外部基表不能存在任何约束条件；

6．外部基表不能为临时表，不能建立分区；

7．外部基表上不能建立任何索引；

8．外部基表是只读的，不存在表锁，不允许任何针对外部表的增删改数据操作，不允许 TRUNCATE 外部表操作；

9．控制文件路径，以及数据文件路径建议采用绝对路径；

10．控制文件的书写格式为：

```
[OPTIONS(
<id>=<value>
……
    )]
LOAD [DATA]
INFILE [<file_option>|<directory_option>]
[BADFILE <path_name>]
<into_table_clause>

<file_option>::= [LIST] <file_option 子句> [,<file_option 子句>]
<file_option 子句>::=<path_name> [<row_term_option>]
<path_name> ::=文件地址
<row_term_option> ::=STR [X] <delimiter>

<directory_option> ::= DIRECTORY <path_name> [<row_term_option>]

<into_table_clause> ::= <into_table_single>{<into_table_single>}
<into_table_single> ::=INTO TABLE [<schema>.]<tablename>
                         [FIELDS [TERMINATED BY] [X] <delimiter>]
<schema> ::=模式名
<tablename> ::=表名
<delimiter> ::='<字符串常量>'
```

其中OPTIONS选项为可选部分，目前OPTIONS中支持DATA、LOG、ERRORS、BADFILE、NULL_STR、SKIP、CHARACTER_CODE 选项，以逗号（,）分隔，含义和 dmfldr 参数说明相同，详细请参考《DM8_dmfldr 使用手册》。

11. 如果没有使用<参数选项>的 RECORDS 指定行分隔符，则在数据文件中的一行数据必须以回车结束；

12. 外部表支持查询 ROWID、USER 和 UID 伪列，不支持查询 TRXID 伪列。

举例说明

例 1 指定操作系统的一个文本文件作为数据文件，编写控制文件及建表语句。

编写数据文件（f:\ext_table\data.txt），首行数据如下：

```
a|abc|varchar_data|12.34|12.34|12.34|12.34|0|1|1|1234|1234|1234|100|11|12
34|1|1|14.2|12.1|12.1|1999-10-01|9:10:21|2002-12-12|15
```

编写控制文件（f:\ext_table\ctrl.txt）如下：

```
LOAD DATA
INFILE  'f:\ext_table\data.txt'
INTO TABLE EXT
FIELDS '|'
```

建表语句：

```
DROP TABLE EXT;
CREATE EXTERNAL TABLE EXT (
L_CHAR CHAR(1),
L_CHARACTER CHARACTER(3),
L_VARCHAR VARCHAR(20),
L_NUMERIC NUMERIC(6,2),
L_DECIMAL DECIMAL(6,2),
```

```
    L_DEC DEC(6,2),
    L_MONEY DECIMAL(19,4),
    L_BIT BIT,
    L_BOOL BIT,
    L_BOOLEAN BIT,
    L_INTEGER INTEGER,
    L_INT INT,
    L_BIGINT BIGINT,
    L_TINYINT TINYINT,
    L_BYTE BYTE,
    L_SMALLINT SMALLINT,
    L_BINARY BINARY,
    L_VARBINARY VARBINARY,
    L_FLOAT FLOAT,
    L_DOUBLE DOUBLE,
    L_REAL REAL,
    L_DATE DATE,
    L_TIME TIME,
    L_TIMESTAMP TIMESTAMP,
    L_INTERVAL INTERVAL YEAR
    )FROM 'f:\ext_table\ctrl.txt';
```

系统执行建表语句后，就在数据库中建立了相应的外部基表。查询 ext_table 表：

```
SELECT * FROM EXT;
```

查询结果：

```
    L_CHAR L_CHARACTER L_VARCHAR L_NUMERIC L_DECIMAL L_DEC L_MONEY L_BIT
L_BOOL L_BOOLEAN L_INTEGER L_INT L_BIGINT L_TINYINT L_BYTE L_SMALLINT
L_BINARY L_VARBINARY L_FLOAT L_DOUBLE L_REAL L_DATE L_TIME L_TIMESTAMP
L_INTERVAL a abc varchar_data 12.34 12.34 12.34 12.3400 0 1 1 1234
1234 1234 100 11 1234 0x01 0x01 14.2 12.1 12.1 1999-10-01 09:10:21
2002-12-12 00:00:00.0 INTERVAL '15' YEAR(2)
```

例 2 指定操作系统的一个文本文件作为数据文件（d:\data.txt），数据如下：

```
10|9|7
4|3|2|5
```

建表语句：

```
    create external table ext_table2(c1 int,c2 int,c3 int) from datafile
'd:\data.txt' parms(fields delimited by '|', records delimited by 0x0d0a);
```

查询结果：

```
select * from ext_table2;
行号       C1          C2          C3
---------- ----------- ----------- -----------
1          10          9           7
2          4           3           2
```

例 3 编写控制文件（F:\TESTEXTERNTABLE\quan.ctrl），内容如下：

```
OPTIONS(
```

```
DATA = 'F:\TESTEXTERNTABLE\quan.txt'
ERRORS                    =   5
 BADFILE                  =    'F:\TESTEXTERNTABLE\t1.bad'
 LOG                      =  'F:\TESTEXTERNTABLE\t1.log'
 NULL_STR                 = ' ffff '
 SKIP                     =    0
 CHARACTER_CODE =    'utf-8'
)
LOAD DATA
INFILE 'F:\TESTEXTERNTABLE\quan.txt' STR x '0D0A'
BADFILE 'F:\TESTEXTERNTABLE\test1.bad'
INTO TABLE fldr1
FIELDS TERMINATED BY  '||'
```

编写数据文件（F:\TESTEXTERNTABLE\quan.txt）如下：

```
1||ab||2
1||ab||1
```

建表语句：

```
CREATE EXTERNAL TABLE  fldr1
(
"C1" NUMBER(2,1),
"C2" VARCHAR(4),
"C3" NUMBER(2,0))
FROM     'F:\TESTEXTERNTABLE\quan.ctrl';
```

查询表 fldr1 中的数据：

```
select * from fldr1;
行号       C1  C2  C3
---------- --- -- --
1          1.0 ab 2
2          1.0 ab 1
```

3.6.1.3 定义 HUGE 表

语法格式

```
CREATE HUGE TABLE <表名定义> <表结构定义>[<PARTITION 子句>] [<STORAGE 子句 1>][<压缩子句>] [<日志属性>][<DISTRIBUTE 子句>];
<表名定义> ::= [<模式名>.] <表名>
<表结构定义> ::=<表结构定义 1> | <表结构定义 2>
<表结构定义 1> ::= (<列定义> {,<列定义>} [<表级约束定义>{,<表级约束定义>}])
<表结构定义 2> ::= AS <不带 INTO 的 SELECT 语句>[<DISTRIBUTE 子句>]
<列定义> ::=<列名> <数据类型>[DEFAULT<列缺省值表达式>][<列级约束定义>][<STORAGE 子句 2>][<存储加密子句>]
<表级约束定义> ::= [CONSTRAINT <约束名>] <表级完整性约束>
<表级完整性约束> ::=
<唯一性约束选项> (<列名> {,<列名>}) [USING INDEX TABLESPACE <表空间名> | DEFAULT]|
FOREIGN KEY (<列名>{,<列名>}) <引用约束> |
```

```
    CHECK (<检验条件>)
    <列级约束定义> ::= <列级完整性约束>{,<列级完整性约束>}
    <列级完整性约束> ::= [CONSTRAINT <约束名>] [NOT] NULL |<唯一性约束选项> [USING
INDEX TABLESPACE <表空间名> | DEFAULT]
    <唯一性约束选项> ::= [PRIMARY KEY] |UNIQUE
    <存储加密子句> ::= <存储加密子句 1>|<存储加密子句 2>
    <存储加密子句 1> ::= ENCRYPT [<加密用法>|<加密用法><加密模式>|<加密模式>]
    <存储加密子句 2> ::= ENCRYPT { <加密用法>|<加密用法><加密模式>|<加密模式>}<散列选项>
    <加密用法> ::= WITH <加密算法>
    <加密模式> ::= <透明加密模式> | <半透明加密模式>
    <透明加密模式> ::= AUTO [<加密密码 1> | <加密密码 2>]
    <加密密码 1> ::= BY <口令>
    <加密密码 2> ::= BY  WRAPPED '<口令密文>'
    <半透明加密模式> ::= MANUAL
    <散列选项> ::= HASH WITH [<密码引擎名>.]<散列算法> [<加盐选项>]
    <加盐选项> ::= [NO] SALT
    <加密算法> ::= <DES_ECB | DES_CBC | DES_CFB|DES_OFB|DESEDE_ECB|
                DESEDE_CBC | DESEDE_CFB|DESEDE_OFB | AES128_ECB |
                AES128_CBC | AES128_CFB | AES128_OFB | AES192_ECB |
                AES192_CBC | AES192_CFB | AES192_OFB | AES256_ECB |
                AES256_CBC | AES256_CFB | AES256_OFB | RC4>
    <散列算法> ::= <MD5 | SHA1 | SHA224 | SHA256 | SHA384 | SHA512>
    <PARTITION 子句> 参见 3.6.1.1 节
    <STORAGE 子句 1> ::= STORAGE(<STORAGE1 子项>,{<STORAGE1 子项>,} ON <HTS 表空间名>)
    <STORAGE1 子项>::=
               [SECTION (<区大小>)]|
               [FILESIZE (<文件大小>)]|
               [STAT [NONE| SYNCHRONOUS | ASYNCHRONOUS][ON | EXCEPT ( col_lst )] ]|
               [<WITH|WITHOUT> DELTA]
    <STORAGE 子句 2> ::= STORAGE(STAT NONE)
    <压缩子句> ::=
        COMPRESS [LEVEL <压缩级别>] [FOR 'QUERY [LOW | HIGH]']|
        COMPRESS  [LEVEL <压缩级别>] [FOR 'QUERY [LOW | HIGH]'] (<列名> [LEVEL <
压缩级别>]  [FOR 'QUERY [LOW | HIGH]'] {,<列名> [LEVEL <压缩级别>] [FOR 'QUERY [LOW
| HIGH]'] })|
        COMPRESS  [LEVEL <压缩级别>]  [FOR 'QUERY [LOW | HIGH]'] EXCEPT (<列名>{,<
列名>})
    <DISTRIBUTE 子句> ::=
        DISTRIBUTED[<RANDOMLY>|<FULLY>]|
        DISTRIBUTED BY [<HASH>](<列名> {,<列名>})|
        DISTRIBUTED BY RANGE (<列名> {,<列名>})(<范围分布项> {,<范围分布项>})|
        DISTRIBUTED BY LIST (DEFAULT|<<列名> {,<列名>}>)(<列表分布项> {,<列表分布
项>})
   <范围分布项> ::=
```

```
        VALUES LESS THAN (<表达式>{,<表达式>}) ON <实例名>|
        VALUES EQU OR LESS THAN (<表达式>{,<表达式>}) ON <实例名>
    <列表分布项> ::= VALUES (<表达式>{,<表达式>}) ON <实例名>
    <日志属性> ::=
        LOG NONE|
        LOG LAST|
        LOG ALL
```

参数

1. <表名> 指明被创建的 HUGE 表名。普通 HUGE 表，由于表名+辅助表名最大长度不大于 128 字节，则表名不大于 123 字节；分区 HUGE 表，由于子表名+辅助表名的长度不大于 128 字节，则子表名不大于 123 字节；

2. <区大小> 指一个区的数据行数。区的大小必须是 2 的 n 次方，如果不是则向上对齐。取值范围：1024 行~1024*1024 行。不指定则默认值为 65536 行；

3. <storage 子句 1>中 STAT [NONE| SYNCHRONOUS | ASYNCHRONOUS] [ON | EXCEPT (col_lst)] 设置 HUGE 表的统计状态。

   1) 设置表的统计状态[NONE| SYNCHRONOUS | ASYNCHRONOUS]
      a) NONE：不计算统计信息，在数据修改时不做数据的统计。
      b) SYNCHRONOUS：实时计算统计信息。
      c) ASYNCHRONOUS：异步计算统计信息，仅支持 WITH DELTA 的 HUGE 表。
      d) 省略此项：如果是 WITH DELTA 的 HUGE 表，统计状态参考 INI 参数 HUGE_STAT_MODE；如果是 WITHOUT DELTA 为实时统计。
   2) 设置列[<ON | EXCEPT> (col_lst)]，用于设置表中特定某些列的统计信息状态
      a) 如果列设置省略，则表示所有列上的统计开关均为打开。
      b) 列设置仅对计算统计信息时有效；如果表的统计状态是 NONE 状态，此时设置列，则报错。
      c) 如果是 ON，则对列表中的列计算统计信息，其余的不计算统计信息。
      d) 如果是 EXCEPT，则对列表中的列不计算统计信息，其余的计算统计信息。
      e) 不支持同时在表和列的定义中指定列的统计信息状态，如果在表定义<storage 子句 1>的 stat 子句中指定了列，在表中某些列定义的<storage 子句 2>中也指定了 stat 子句，则报错。
      f) 允许下列情况：表设置了要计算统计信息，但所有列上都设置不计算统计信息。

4. <storage 子句 2>的 STAT NONE 对某一列进行设置，该列不计算统计信息，在修改时不做数据的统计；此处不设置 STAT NONE，表示进行统计；

5. <HTS 表空间名> 指要创建的 HUGE 表所属的 HTS 表空间。不指定则存储于默认系统 HUGE 表空间 HMAIN 中；

6. <文件大小> 指创建 HUGE 表时指定的单个文件的大小。取值范围为 16M~1024*1024M。文件大小必须是 2 的 n 次方，如果不是则向上对齐。不指定则默认为 64M；

7. <压缩级别> 为特定列指定通用压缩级别，有效值范围为：0~10，分别代表不同的压缩算法和压缩级别。有两种压缩算法：SNAPPY 和 ZIP。10 采用 SNAPPY 算法轻量级方式压缩。2~9 采用 ZIP 算法压缩，2~9 代表压缩级别，值越小表示压缩比越低、压缩速率越快；值越大表示压缩比越高、压缩速度越慢。0 和 1 为快捷使用，默认值为 0。0 等价

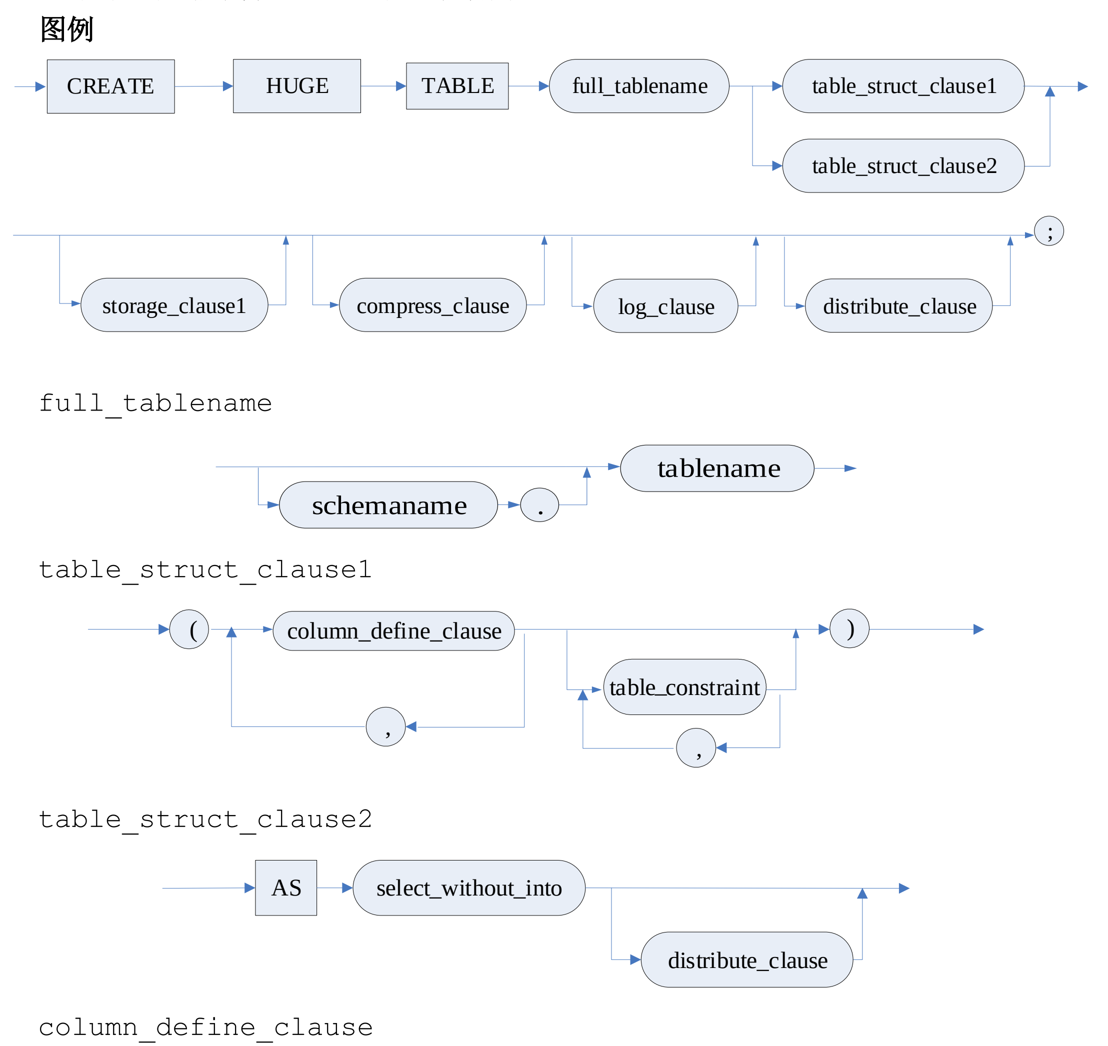
于 LEVEL 2；1 等价于 LEVEL 9；

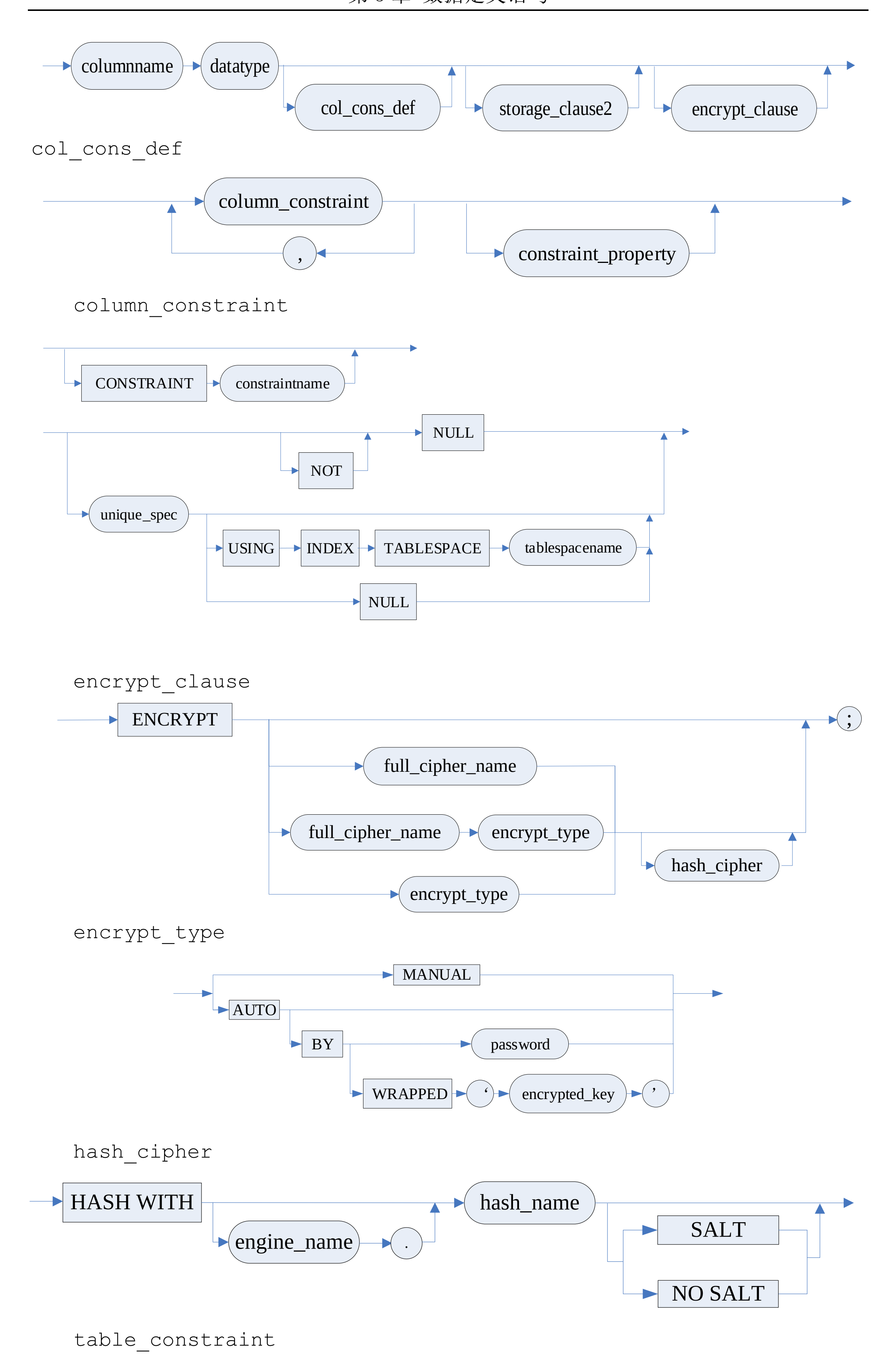
8． FOR 'QUERY [LOW]'表示进行规则压缩；FOR 'QUERY HIGH'表示结合进行规则压缩与通用压缩结合，前者的压缩比一般在 1:1 至 1:3 之间，后者一般为 1:3 至 1:5 之间。规则压缩方式一般适用于具有一定的数据规则的数据的压缩，例如重复值较多等。若某列的类型为字符串类型且定义长度超过 48，则即使指定规则压缩也无效，实际只进行通用压缩；

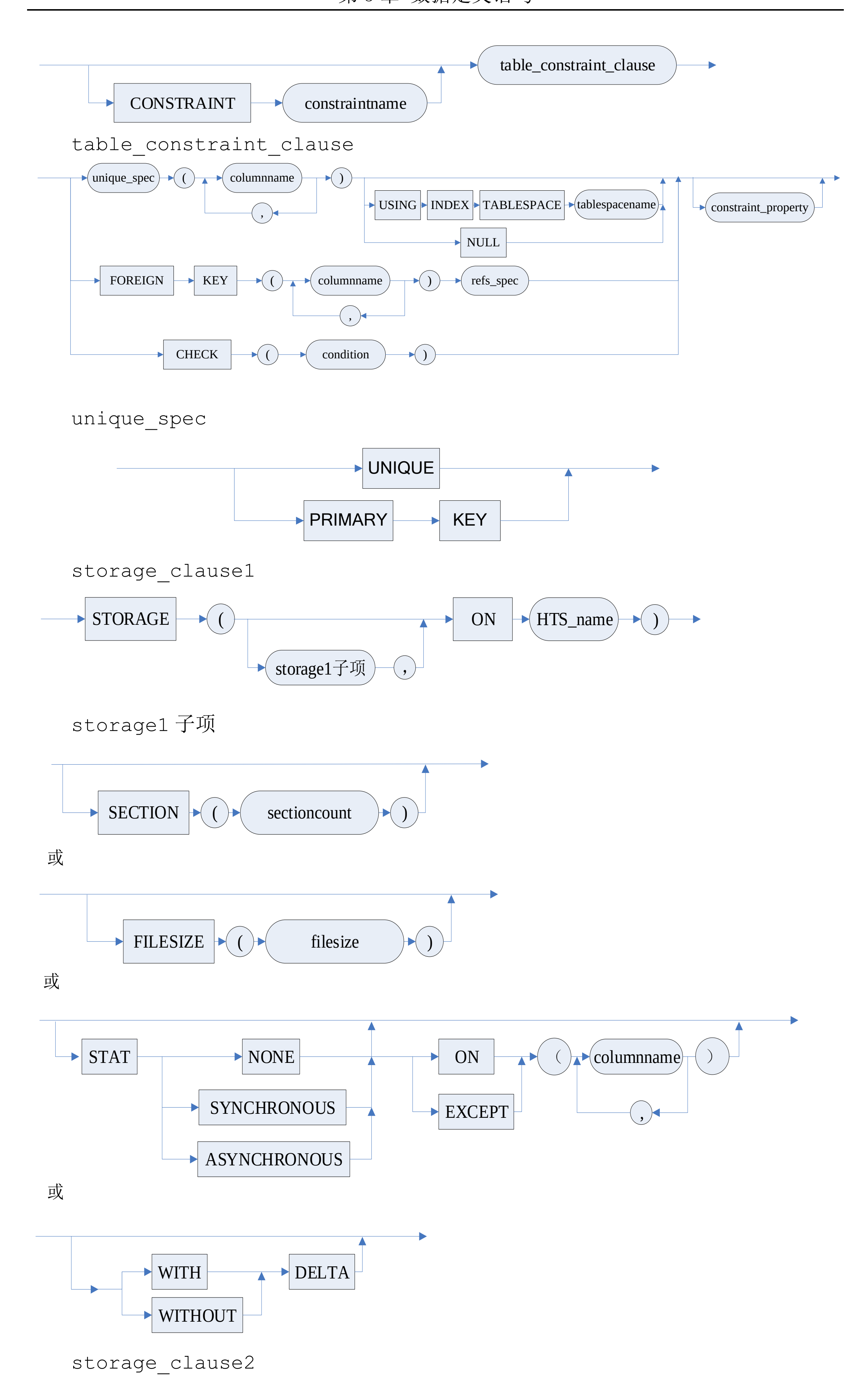
9． <日志属性> 此属性仅对非事务型 HUGE 表有效，支持通过做日志来保证数据的完整性。完整性保证策略主要是通过数据的镜像来实现的，镜像的不同程度可以实现不同程度的完整性恢复。三种选择：1）LOG NONE：不做镜像。相当于不做数据一致性的保证，如果出错只能手动通过系统函数 SF_REPAIR_HFS_TABLE(模式名,表名)来修复表数据。2）LOG LAST：做部分镜像。但是在任何时候都只对当前操作的区做镜像，如果当前区的操作完成了，这个镜像也就失效了，并且可能会被下一个被操作区覆盖，这样做的好处是镜像文件不会太大，同时也可以保证数据是完整的。但有可能遇到的问题是：一次操作很多的情况下，有可能一部分数据已经完成，另一部分数据还没有来得及做的问题。3）LOG ALL：全部做镜像。在操作过程中，所有被修改的区都会被记录下来，当一次操作修改的数据过多时，镜像文件有可能会很大，但能够保证操作完整性。默认选择为 LOG LAST；

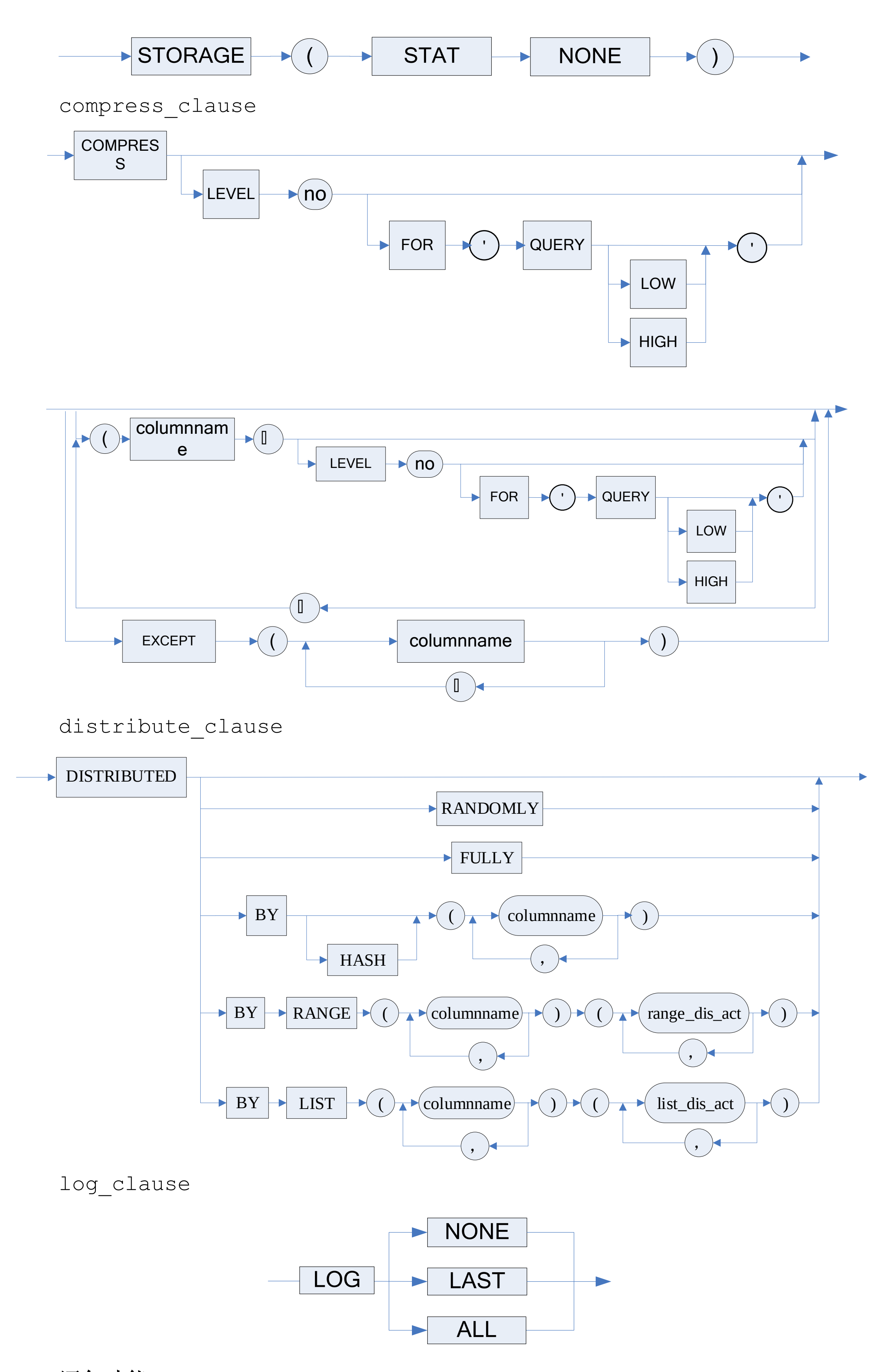
10．<WITH|WITHOUT> DELTA WITH DELTA 表示创建事务型 HUGE 表；WITHOUT DELTA 表示创建非事务型 HUGE 表，缺省为 WITHOUT DELTA。

图例




columnname
datatype
col_cons_def
storage_clause2
encrypt_clause
col_cons_def
column_constraint
,
constraint_property
column_constraint
CONSTRAINT
constraintname
NULL
NOT
unique_spec
USING
INDEX
TABLESPACE
tablespacename
NULL
encrypt_clause
ENCRYPT
;
full_cipher_name
full_cipher_name
encrypt_type
hash_cipher
encrypt_type
encrypt_type
MANUAL
AUTO
BY
password
WRAPPED
‘
encrypted_key
’
hash_cipher
HASH WITH
engine_name
.
hash_name
SALT
NO SALT
table_constraint


CONSTRAINT
constraintname
table_constraint_clause
table_constraint_clause
unique_spec
(
columnname
,
)
USING
INDEX
TABLESPACE
tablespacename
NULL
constraint_property
FOREIGN
KEY
(
columnname
,
)
refs_spec
CHECK
(
condition
)
unique_spec
UNIQUE
PRIMARY
KEY
storage_clause1
STORAGE
(
storage1子项
,
ON
HTS_name
)
storage1 子项
SECTION
(
sectioncount
)
或
FILESIZE
(
filesize
)
或
STAT
NONE
SYNCHRONOUS
ASYNCHRONOUS
ON
EXCEPT
(
columnname
,
)
或
WITH
WITHOUT
DELTA
storage_clause2



语句功能

供 DBA 或具有 CREATE HUGE TABLE 权限的用户创建 HUGE 表。

使用说明

1．非事务型 HUGE 表的插入、删除与更新操作处理都不能进行回滚；

2．建 HUGE 表时仅支持定义 NULL、NOT NULL、UNIQUE 约束以及 PRIMARY KEY，后两种约束也可以通过 ALTER TABLE 的方式添加，但这两种约束不检查唯一性；

3．HUGE 不允许建立聚簇索引，允许建立二级索引，不支持建位图索引，其中 UNIQUE 索引不检查唯一性；

4．不支持 SPACE LIMIT（空间限制）；

5．不支持建立全文索引；

6．不支持使用自定义类型；

7．不支持引用约束；

8．不支持 IDENTITY 自增列；

9．不支持大字段列；

10．不支持建触发器；

11．不支持游标的修改操作；

12．PK 和 UNIQUE 约束不检查唯一性，对应的索引都为虚索引；UNIQUE 索引也不检查唯一性，为实索引，索引标记中不包含唯一性标记，即和普通二级索引相同；

13．不允许对分区子表设置 SECTION 和 WITH/WITHOUT DELTA；

14．对于非事务型 HUGE 表，若指定记录区统计信息，可能因为统计信息超长造成记录插入或更新失败；

15．HUGE 表备份还原请参考附录 3。

举例说明

例 以 SYSDBA 身份登录数据库后，创建 HUGE 表 orders。

```
CREATE HUGE TABLE orders
(
    o_orderkey          INT,
    o_custkey           INT,
    o_orderstatus           CHAR(1),
    o_totalprice            FLOAT,
    o_orderdate         DATE,
    o_orderpriority         CHAR(15),
    o_clerk             CHAR(15),
    o_shippriority          INT,
    o_comment           VARCHAR(79) STORAGE(stat none)
)STORAGE(SECTION(65536) , FILESIZE(64), WITH DELTA, ON HTS_NAME) COMPRESS
LEVEL 9 FOR 'QUERY HIGH'  (o_comment);
```

这个例子创建了一个名为 ORDERS 的事务型 HUGE 表，ORDERS 表的区大小为 65536 行，文件大小为 64M，指定所在的表空间为 HTS_NAME，o_comment 列指定的区大小为不做统计信息，其它列（默认）都做统计信息，指定列 o_comment 列压缩类型为查询高压缩率，压缩级别为 9。

3.6.1.4 定义水平分区表

水平分区包括范围分区、哈希分区和列表分区三种。水平分区表的创建需要通过<PARTITION 子句>指定。

范围（RANGE）分区，按照分区列的数据范围，确定实际数据存放位置的划分方式。

列表（LIST）分区，通过指定表中的某一个列的离散值集，来确定应当存储在一起的

数据。范围分区是按照某个列上的数据范围进行分区的，如果某个列上的数据无法通过划分范围的方法进行分区，并且该列上的数据是相对固定的一些值，可以考虑使用 LIST 分区。一般来说，对于数字型或者日期型的数据，适合采用范围分区的方法；而对于字符型数据，取值比较固定的，则适合于采用 LIST 分区的方法。

哈希（HASH）分区，对分区列值进行 HASH 运算后，确定实际数据存放位置的划分方式，主要用来确保数据在预先确定数目的分区中平均分布，允许只建立一个 HASH 分区。在很多情况下，用户无法预测某个列上的数据变化范围，因而无法实现创建固定数量的范围分区或 LIST 分区。在这种情况下，DM 哈希分区提供了一种在指定数量的分区中均等地划分数据的方法， 基于分区键的散列值（HASH 值）将行映射到分区中。当用户向表中写入数据时，数据库服务器将根据一个哈希函数对数据进行计算，把数据均匀地分布在各个分区中。在哈希分区中，用户无法预测数据将被写入哪个分区中。

在很多情况下，经过一次分区并不能精确地对数据进行分类，这时需要多级分区表。在进行多级分区的时候，三种分区类型还可以交叉使用。

语法格式

```
CREATE TABLE <表名定义> <表结构定义>;
<表名定义> ::= [<模式名>.] <表名>
<表结构定义>::= (<列定义> {,<列定义>} [,<表级约束定义>{,<表级约束定义>}]) [ON COMMIT <DELETE | PRESERVE> ROWS] <PARTITION 子句>[<空间限制子句>] [<STORAGE 子句>][<压缩子句>] [<ROW MOVEMENT 子句>][<add_log 子句>][<DISTRIBUTE 子句>]
<列定义>、<表级约束定义>、[ON COMMIT <DELETE | PRESERVE> ROWS]、<空间限制子句>、<STORAGE 子句>、<压缩子句>、<DISTRIBUTE 子句>的语法，参见 3.6.1 节说明
<PARTITION 子句> ::= PARTITION BY <PARTITION 项>
<PARTITION 项> ::=
    RANGE (<列名>{,<列名>}) [INTERVAL <间隔表达式>] [<子分区模板>{,<子分区模板>}](<RANGE 分区项> {,<RANGE 分区项>}) |
    HASH (<列名>{,<列名>}) [<子分区模板>{,<子分区模板>}]PARTITIONS <分区数>[<STORAGE HASH 子句>]|
    HASH(<列名>{,<列名>})[<子分区模板>{,<子分区模板>}] (<HASH 分区项> {,<HASH 分区项>})|
    LIST(<列名>)[<子分区模板>{,<子分区模板>}](<LIST 分区项> {,<LIST 分区项>})
<RANGE 分区项>::= PARTITION <分区名> VALUES [EQU OR] LESS THAN (<常量表达式|<日期函数表达式>|MAXVALUE>{,<常量表达式|<日期函数表达式>|MAXVALUE>}) [<STORAGE 子句>][<子分区描述项>]
<日期函数表达式> ::= <to_date 函数表达式> | <to_datetime 函数表达式> | <to_timestamp 函数表达式>
<HASH 分区项>::= PARTITION <分区名> [<STORAGE 子句>][<子分区描述项>]
<LIST 分区项>::= PARTITION <分区名> VALUES (DEFAULT|<<表达式>,{<表达式>}>) [<STORAGE 子句>][<子分区描述项>]
<子分区描述项> ::=
    (<RANGE 子分区描述项>{,<RANGE 子分区描述项>})|
    (<HASH 子分区描述项>{,<HASH 子分区描述项>})|
    SUBPARTITIONS <分区数> [<STORAGE HASH 子句>]|
    (<LIST 子分区描述项>{,<LIST 子分区描述项>})
<RANGE 子分区描述项> ::= <RANGE 子分区项>[<子分区描述项>]
```

```
<HASH 子分区描述项> ::= <HASH 子分区项>[<子分区描述项>]
<LIST 子分区描述项> ::= <LIST 子分区项>[<子分区描述项>]
<RANGE 子分区项> ::=
SUBPARTITION <分区名> VALUES [EQU OR] LESS THAN (<常量表达式|<日期函数表达式>|MAXVALUE>){,<常量表达式|<日期函数表达式>|MAXVALUE>})
<HASH 子分区项> ::= SUBPARTITION <分区名>
<LIST 子分区项> ::= SUBPARTITION <分区名> VALUES (DEFAULT|<<表达式>,{<表达式>}>)
<间隔表达式> ::= <日期间隔函数> | <数值常量>
<子分区模板> ::= <RANGE 子分区模板>|<HASH 子分区模板>|<LIST 子分区模板>
<RANGE 子分区模板> ::= SUBPARTITION BY RANGE (<列名>{,<列名>})[SUBPARTITION TEMPLATE (<RANGE 分区项> {,<RANGE 分区项>})]
<HASH 子分区模板> ::=
    SUBPARTITION BY HASH (<列名>{,<列名>})SUBPARTITION TEMPLATE SUBPARTIONS <分区数> [<STORAGE HASH 子句>]|
    SUBPARTITION BY HASH (<列名>{,<列名>})SUBPARTITION TEMPLATE (<HASH 分区项> {,<HASH 分区项>})
<LIST 子分区模板> ::= SUBPARTITION BY LIST (<列名>{,<列名>})[SUBPARTITION TEMPLATE (<LIST 分区项> {,<LIST 分区项>})]
<STORAGE HASH 子句> ::= STORE IN (<表空间名列表>)
<ROW MOVEMENT 子句> ::=
    ENABLE ROW MOVEMENT |
    DISABLE ROW MOVEMENT
```

参数

1. <模式名> 指明该表属于哪个模式，缺省为当前模式；

2. <表名> 指明被创建的基表名，基表名最大长度 128 字节；如果是分区表，主表名和分区名遵循“主表名_分区名”总长度最大不超过 128 字节。按数目方式创建 HASH 分区，子分区命名规则为“DMHASHPART+分区序号”， 其他方式的分区名由用户指定；

3. <子分区描述项> 多级分区表支持自定义子分区描述项，自定义的子分区描述项分区类型与分区列必须与子分区模板一致。如果子分区模板和自定义子分区描述项均指定了分区信息则按自定义分区描述项处理；

4. <STORAGE HASH 子句> 指定哈希分区依次使用的表空间；

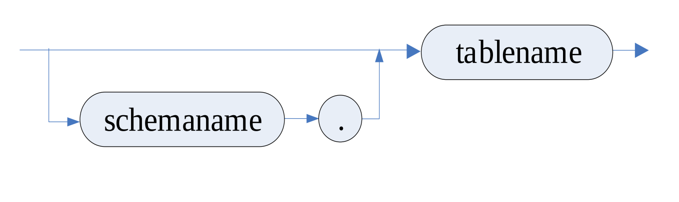
5. <ROW MOVEMENT 子句> 设置行迁移功能，仅对行存储的水平分区表有效，其它表类型自动忽略。ENABLE ROW MOVEMENT 打开行迁移，允许更新后数据发生跨分区的移动。DISABLE ROW MOVEMENT，关闭行迁移，不允许更新后数据发生跨分区的移动。缺省为 DISABLE ROW MOVEMENT；

6. <间隔表达式>中日期间隔函数为 NUMTOYMINTERVAL、NUMTODSINTERVAL；数值常量为：整型、DEC 类型。间隔分区表只能包含一个分区列。

图例



full_tablename



table_struct_clause

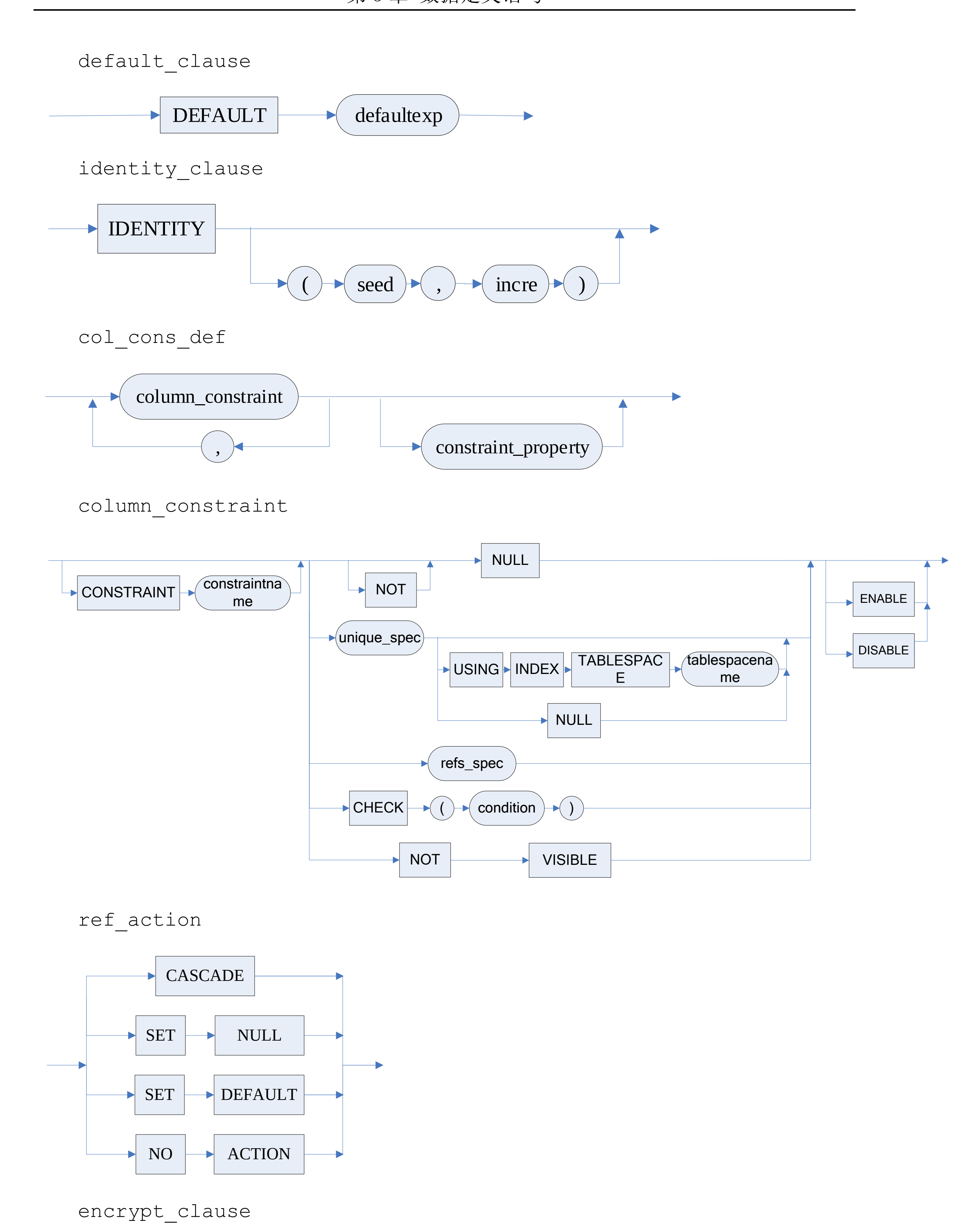
(
column_define_clause
,
,
table_constraint
)
ON
COMMIT
DELETE
PRESERVE
ROWS
partition_clause
spacelimit_clause
storage_clause
compress_clause
row_movement_clause
distribute_clause

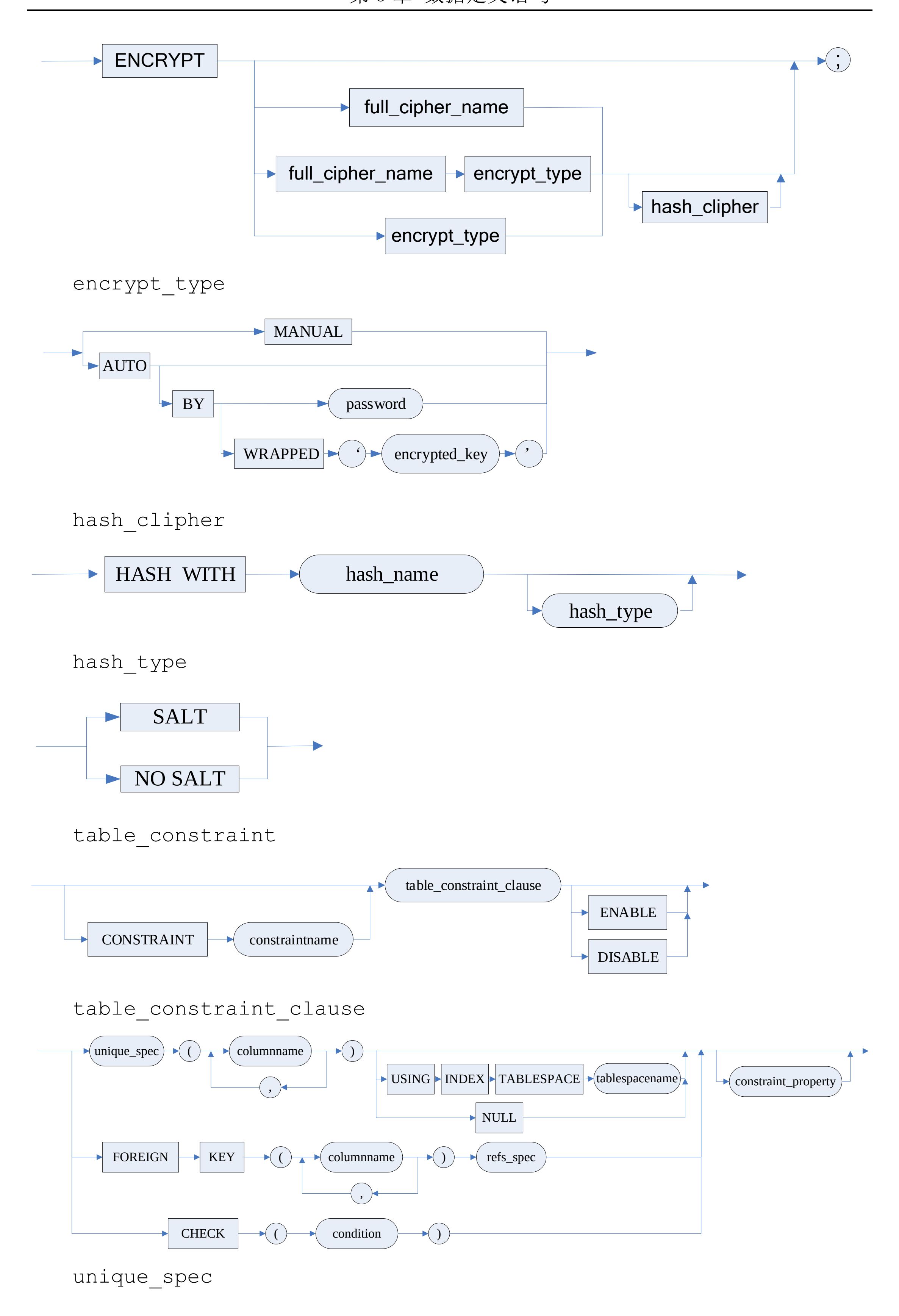
column_define_clause

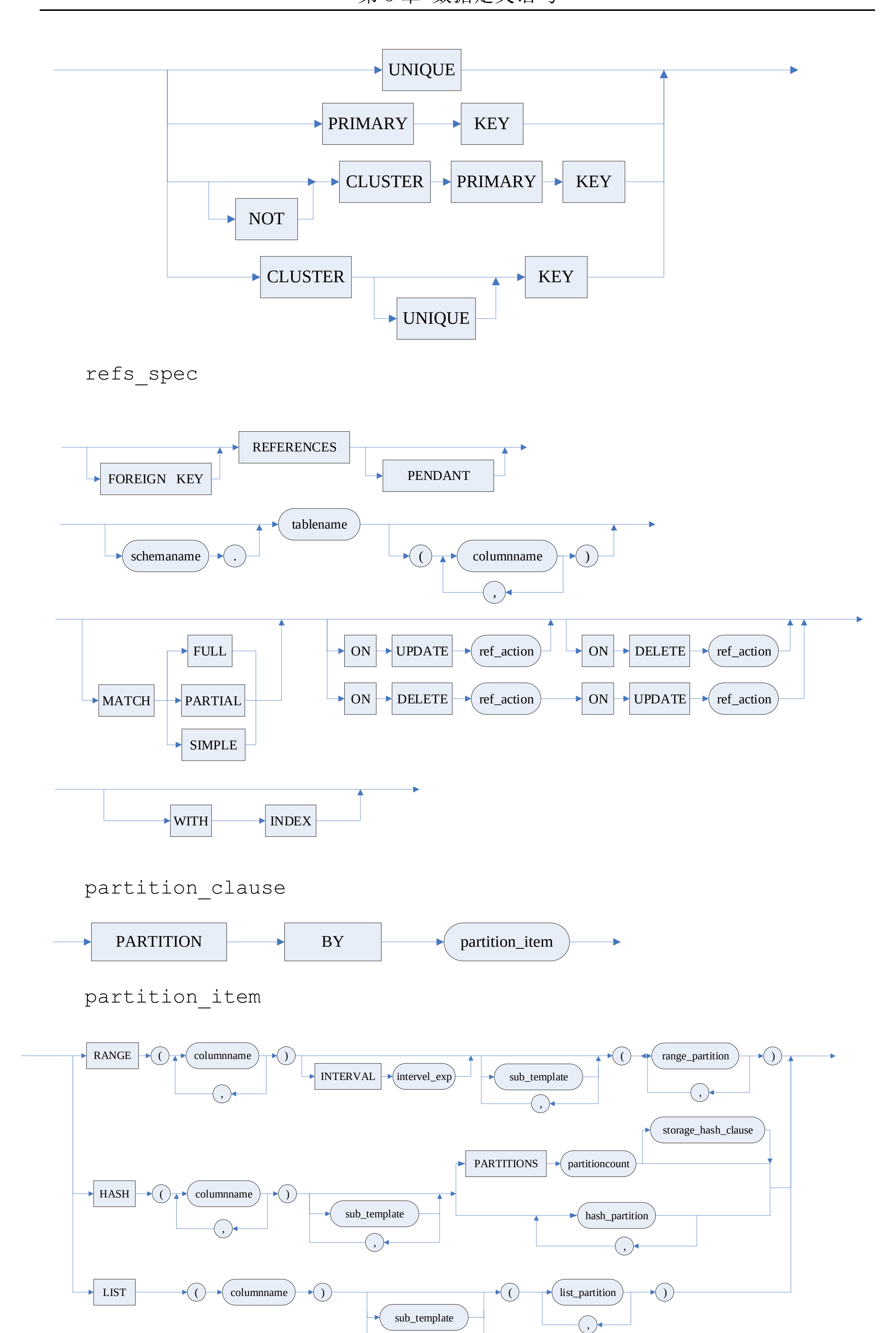
col_define_low
default_clause
col_cons_def
identity_clause
col_cons_def
col_cons_def
default_clause
identity_clause
storage_clause
encrypt_clause

col_define_low

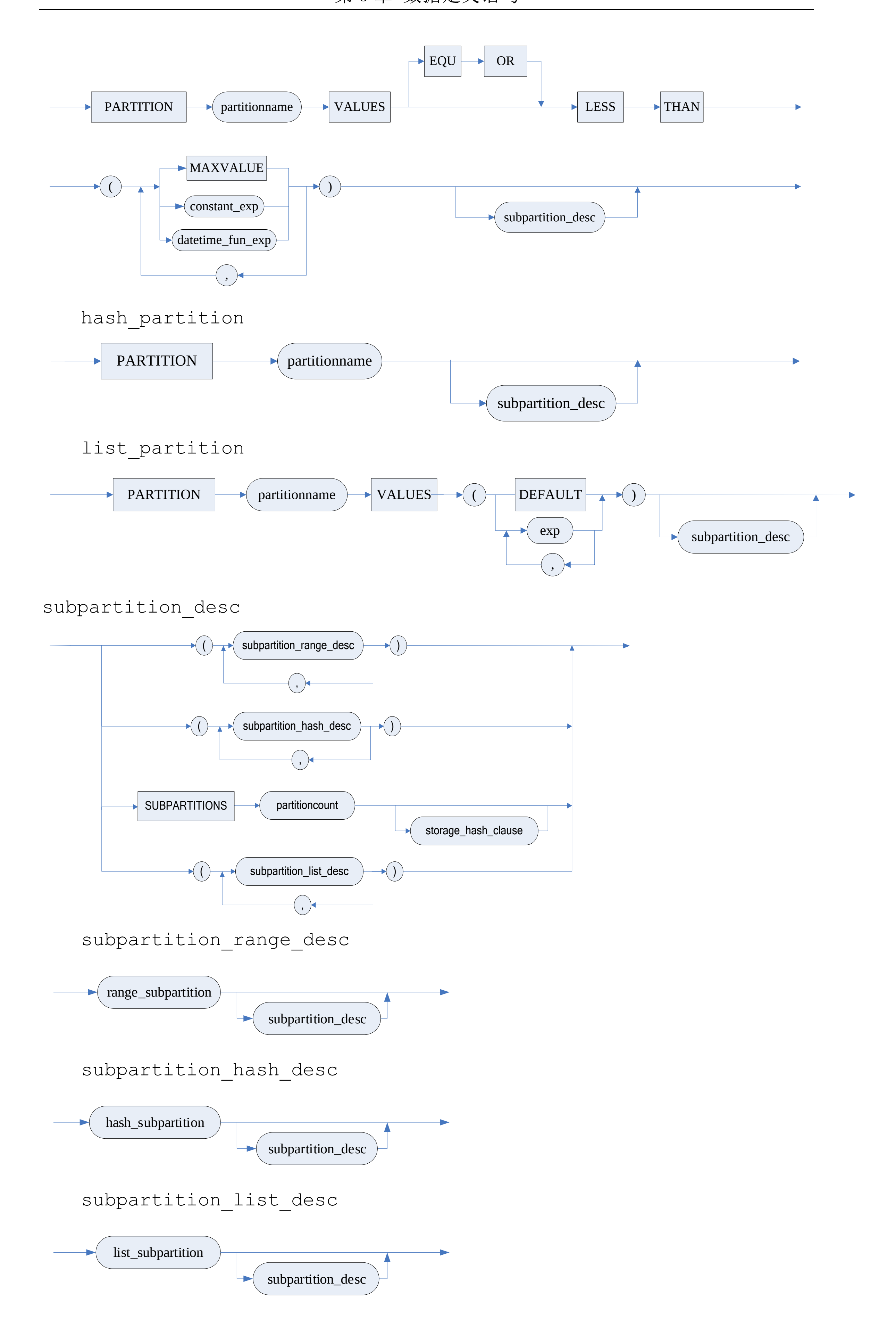
column
datatype
datatype
GENERATED
ALWAYS
AS
(
vir_col_define
)
VIRTUAL
VISIBLE


default_clause
DEFAULT
defaultexp
identity_clause
IDENTITY
(
seed
,
incre
)
col_cons_def
column_constraint
,
constraint_property
column_constraint
CONSTRAINT
constraintname
NOT
NULL
unique_spec
USING
INDEX
TABLESPACE
tablespacename
NULL
refs_spec
CHECK
(
condition
)
NOT
VISIBLE
ENABLE
DISABLE
ref_action
CASCADE
SET
NULL
SET
DEFAULT
NO
ACTION
encrypt_clause

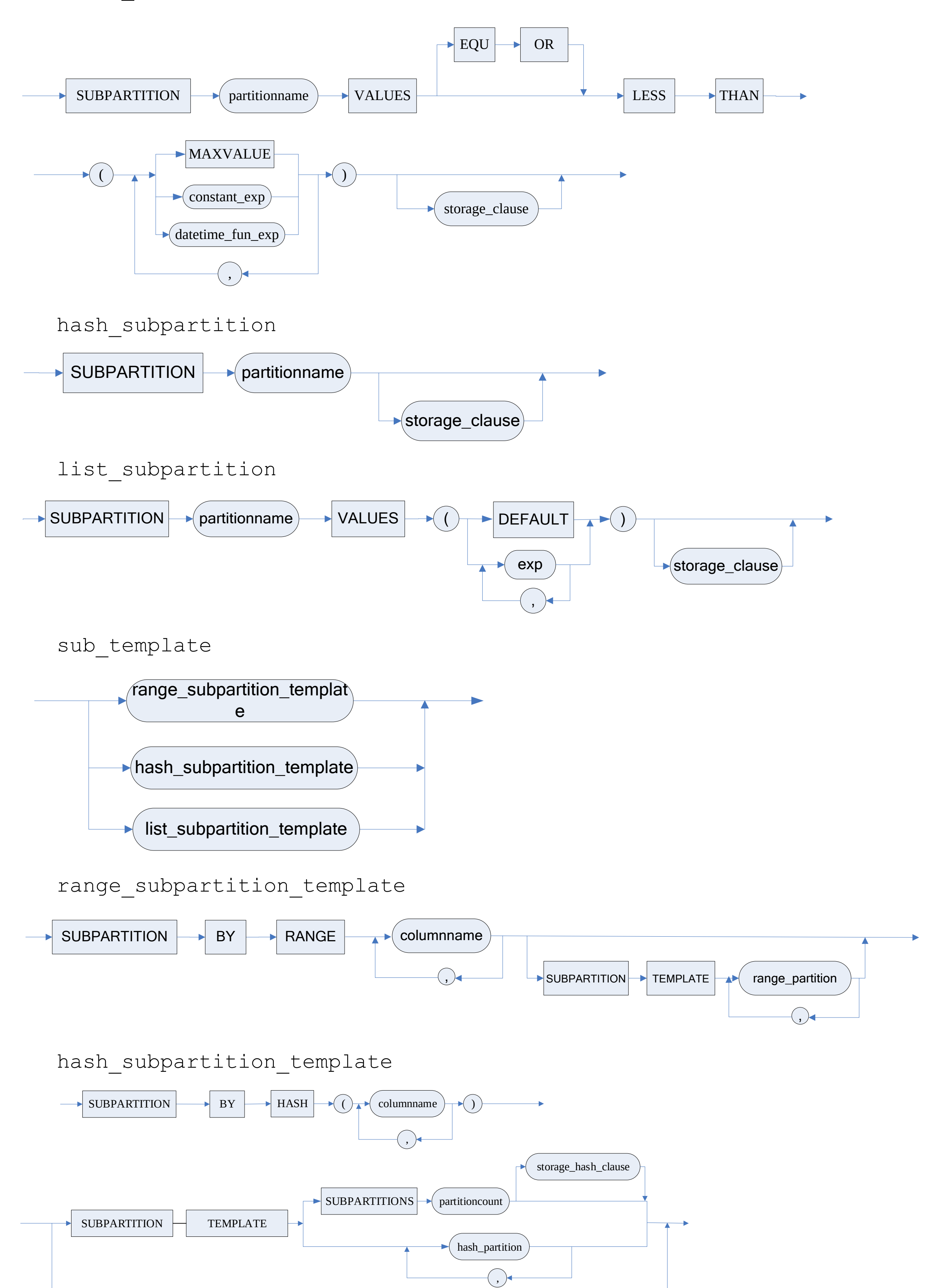

ENCRYPT
;
full_cipher_name
full_cipher_name
encrypt_type
hash_clipher
encrypt_type
encrypt_type
MANUAL
AUTO
BY
password
WRAPPED
‘
encrypted_key
’
hash_clipher
HASH WITH
hash_name
hash_type
hash_type
SALT
NO SALT
table_constraint
table_constraint_clause
CONSTRAINT
constraintname
ENABLE
DISABLE
table_constraint_clause
unique_spec
(
columnname
,
)
USING
INDEX
TABLESPACE
tablespacename
NULL
constraint_property
FOREIGN
KEY
(
columnname
,
)
refs_spec
CHECK
(
condition
)
unique_spec



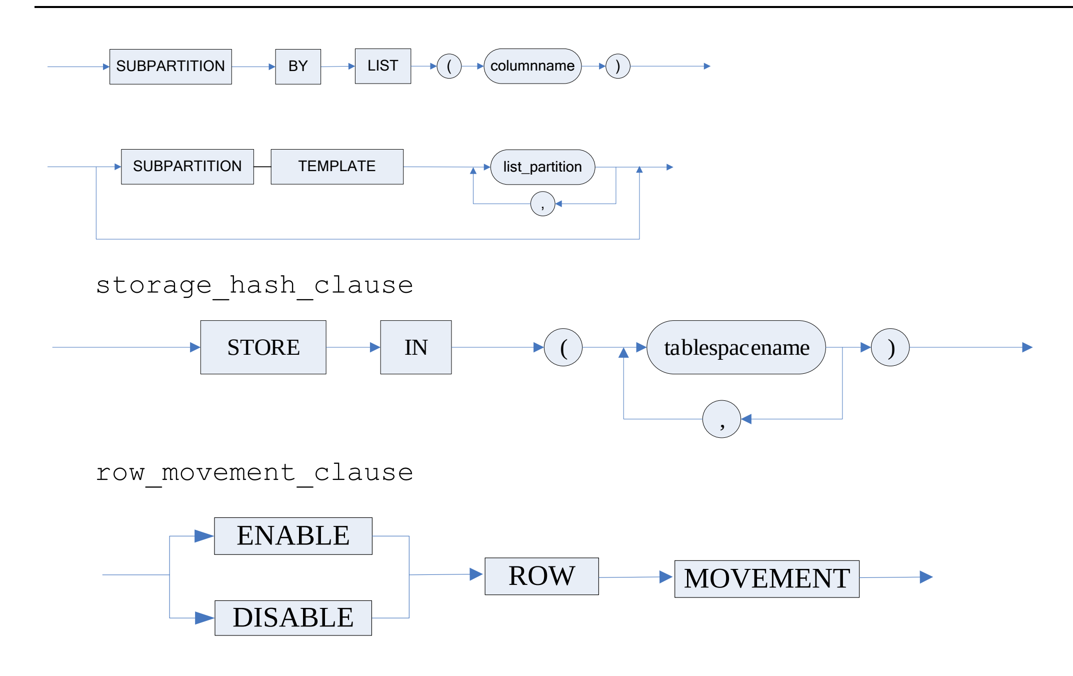
range_partition


PARTITION
partitionname
VALUES
EQU
OR
LESS
THAN
(
MAXVALUE
constant_exp
datetime_fun_exp
,
)
subpartition_desc
hash_partition
PARTITION
partitionname
subpartition_desc
list_partition
PARTITION
partitionname
VALUES
(
DEFAULT
exp
,
)
subpartition_desc
subpartition_desc
(
subpartition_range_desc
,
)
(
subpartition_hash_desc
,
)
SUBPARTITIONS
partitioncount
storage_hash_clause
(
subpartition_list_desc
,
)
subpartition_range_desc
range_subpartition
subpartition_desc
subpartition_hash_desc
hash_subpartition
subpartition_desc
subpartition_list_desc
list_subpartition
subpartition_desc

range_subpartition



list_subpartition_template



语句功能

供 DBA 或具有 CREATE TABLE 权限的用户定义水平分区表。

使用说明

1. <表名>指定所要建立的基表名；

2. <PARTITION 子句>用来指定水平分区。其中 RANGE 和 HASH 可以指定一个或多个列作为分区列，LIST 只能指定一个列作为分区列；

   1) "PARTITION BY RANGE……"子句用来指定范围分区，然后在每个分区中分区列的取值通过 VALUES 子句指定。

   2) "PARTITION BY LIST……"子句用来指定 LIST 分区，然后在每个分区中分区列的取值通过 VALUES 子句指定。当用户向表插入数据时，只要分区列的数据与 VALUES 子句指定的数据之一相等，该行数据便会写入相应的分区子表中。LIST 分区的分区范围值必须唯一，不能重复。

   3) "PARTITION BY HASH……"子句用来指定 HASH 分区。

3. 分区列类型必须是数值型、字符型或日期型，不支持 BLOB、CLOB、IMAGE、TEXT、LONGVARCHAR、BIT、BINARY、VARBINARY、LONGVARBINARY、BFILE、时间间隔类型和用户自定义类型为分区列；

4. 水平分区表指定主键和唯一约束时，分区键必须都包含在主键和唯一约束中，但是全局唯一索引不受此约束；

5. 不能在水平分区表上建立自引用约束；

6. 普通环境中，水平分区表的各级分区数的总和上限是 65535；MPP 环境下，水平分区表的各级分区总数上限取决于 INI 参数 MAX_EP_SITES，上限为 2(16 - log2MAX_EP_SITES)。比如：当 MAX_EP_SITES 为默认值 64 时，分区总数上限为 1024；

7. 可以定义主表的 BRANCH 选项，但不能对水平分区子表进行 BRANCH 项设置，子表的 BRANCH 项只能通过主表继承得到；

8. 水平分区表不支持自增列；

9. 不允许引用水平分区子表作为外键约束；

10. 水平分区子表删除后，会将子表上的数据一起删除；

11. 范围分区和哈希分区的分区键可以多个，最多不超过 16 列；LIST 分区的分区键

必须唯一；

12. 范围分区表使用说明：

    1) 范围分区支持 MAXVALUE 值的使用，MAXVALUE 代表一个比任何值都大的值。MAXVALUE 值需要用户指定才能使用，作为分区中的最大值。
    2) 范围分区的范围值表达式类型应与分区列类型一致，否则按分区列类型进行类型转换。
    3) 对于范围分区，增加分区必须在最后一个分区范围值的后面添加，要想在表的开始范围或中间增加分区，应使用 SPLIT PARTITION 语句。

13. LIST 分区表使用说明：

    1) LIST 分区支持 DEFAULT 关键字的使用，所有不满足分区条件的数据，都划分为 DEFAULT 的分区，但 DEFAULT 关键字需要用户指定，系统不会自动创建 DEFAULT 分区。
    2) LIST 分区范围值不能为 NULL。
    3) LIST 分区子表范围值个数与数据页大小和相关系统表列长度相关，存在以下限制：
        a) 4K 页，单个子表最多支持 120 个范围值。
        b) 8K 页，单个子表最多支持 254 个范围值。
        c) 16K 或 32K 页，单个子表最多支持 270 个范围值。

14. 水平分区表为堆表时，主表及其各子表必须位于同一个表空间。
15. 组合水平分区表层次最多支持八层；
16. 普通表、堆表、列存储表均支持多级分区；
17. 分区子表的存储属性与根表保持一致，忽略子表的 STORAGE 子句。

举例说明

例 1 创建一个范围分区表 callinfo，用来记录用户 2018 年的电话通讯信息，包括主叫号码、被叫号码、通话时间和时长，并且根据季度进行分区。

```
CREATE  TABLE  callinfo(
caller  CHAR(15),
callee  CHAR(15),
time    DATETIME,
duration    INT
)
PARTITION BY RANGE(time)(
PARTITION p1 VALUES LESS THAN ('2018-04-01'),
PARTITION p2 VALUES LESS THAN ('2018-07-01'),
PARTITION p3 VALUES LESS THAN ('2018-10-01'),
PARTITION p4 VALUES EQU OR LESS THAN ('2018-12-31'));
                                    --'2010-12-31'也可替换为MAXVALUE
```

表中的每个分区都可以通过“PARTITION”子句指定一个名称。并且每一个分区都有一个范围，通过“VALUES LESS THAN”子句可以指定上界，而它的下界是前一个分区的上界。如分区 p2 的 time 字段取值范围是['2018-04-01', '2018-07-01')。如果通过“VALUES EQU OR LESS THAN”指定上界，即该分区包含上界值，如分区 p4 的 time 字段取值范围是['2018-10-01', '2018-12-31']。另外，可以对每一个分区指定 STORAGE 子句，不同分区可存储在不同表空间中。

例 2 创建多列分区。创建一个范围分区表 callinfo，以 time，duration 两列为

分区列。

```
CREATE  TABLE  callinfo(
caller  CHAR(15),
callee  CHAR(15),
time    DATETIME,
duration    INT
)
PARTITION BY RANGE(time, duration)(
PARTITION p1 VALUES LESS THAN ('2018-04-01',10),
PARTITION p2 VALUES LESS THAN ('2018-07-01',20),
PARTITION p3 VALUES LESS THAN ('2018-10-01',30),
PARTITION p4 VALUES EQU OR LESS THAN ('2018-12-31', 40) );

insert into CALLINFO values('CHERRY','JACK','2018-12-31',40);
insert into CALLINFO values('CHERRY','JACK','2018-12-31',41); --报错：没有找到合适的分区
```

如果分区表包含多个分区列，采用多列比较方式定位匹配分区。首先，比较第一个分区列值，如果第一列值在范围之内，就以第一列为依据进行分区；如果第一列值处于边界值，那么需要比较第二列的值，根据第二列为依据进行分区；如果第二列的值也处于边界值，需要继续比较后续分区列值，以此类推，直到确定目标分区为止。匹配过程参看下表。

表 3.6.1 多分区列匹配

插入记录	分区范围值		
	(10,10,10)	(20,20,20)	(30,30,30)
(5,100,200)	满足		
(10,10,11)	×	满足	
(20,20,29)	×	×	满足
(31,1,1)	×	×	×

例 3 查询分区子表，直接使用子表名称进行查询。

当在分区表中执行 DML 操作时，实际上是在各个分区子表上透明地修改数据。当执行 SELECT 命令时，可以指定查询某个分区上的数据。

例如，查询 callinfo 表中分区 p1 的数据，可以通过以下方式：

```
SELECT * FROM callinfo PARTITION (p1);
```

例 4 创建一个产品销售记录表 sales，记录产品的销量情况。由于产品只在几个固定的城市销售，所以可以按照销售城市对该表进行 LIST 分区。

```
CREATE  TABLE  sales(
sales_id    INT,
saleman     CHAR(20),
saledate    DATETIME,
city        CHAR(10)
)
PARTITION BY LIST(city)(
PARTITION p1 VALUES ('北京', '天津'),
PARTITION p2 VALUES ('上海', '南京', '杭州'),
```

```
PARTITION p3 VALUES ('武汉', '长沙'),
PARTITION p4 VALUES ('广州', '深圳'),
PARTITION p5 VALUES (default)
);
```

例 5 如果销售城市不是相对固定的，而是遍布全国各地，这时很难对表进行 LIST 分区。如果为该表进行哈希分区，可以很好地解决这个问题。

```
CREATE  TABLE  sales01(
sales_id    INT,
saleman     CHAR(20),
saledate    DATETIME,
city        CHAR(10)
)
PARTITION BY HASH(city)(
PARTITION p1,
PARTITION p2,
PARTITION p3,
PARTITION p4
);
```

如果不指定分区表名，还可以通过指定哈希分区个数来建立哈希分区表。

```
CREATE  TABLE  sales02(
sales_id    INT,
saleman     CHAR(20),
saledate    DATETIME,
city        CHAR(10)
)
PARTITION BY HASH(city)
PARTITIONS 4 STORE IN (ts1, ts2, ts3, ts4);
```

例 6 创建一个产品销售记录表 sales，记录产品的销量情况。由于产品需要按地点和销售时间进行统计，则可以对该表进行多级分区，一级 LIST 分区、二级 RANGE 分区。

```
DROP TABLE SALES;
CREATE  TABLE  SALES(
SALES_ID     INT,
SALEMAN     CHAR(20),
SALEDATE    DATETIME,
CITY          CHAR(10)
)
PARTITION BY LIST(CITY)
 SUBPARTITION BY RANGE(SALEDATE) SUBPARTITION TEMPLATE(
    SUBPARTITION P11 VALUES LESS THAN ('2012-04-01'),
    SUBPARTITION P12 VALUES LESS THAN ('2012-07-01'),
    SUBPARTITION P13 VALUES LESS THAN ('2012-10-01'),
    SUBPARTITION P14 VALUES EQU OR LESS THAN (MAXVALUE))
(
    PARTITION P1 VALUES ('北京', '天津')
```

```
        (
          SUBPARTITION P11_1 VALUES LESS THAN ('2012-10-01'),
          SUBPARTITION P11_2 VALUES EQU OR LESS THAN (MAXVALUE)
        ),
        PARTITION P2 VALUES ('上海', '南京', '杭州'),
        PARTITION P3 VALUES (DEFAULT)
    );
```

在创建多级分区表时，指定了子分区模板，同时子分区 P1 又自定义了子分区描述项 P11_1 和 P11_2。除了 P1 使用自定义的子分区描述项，拥有两个自定义的子分区 P11_1 和 P11_2 之外，其他子分区 P2 和 P3 都使用子分区模板，各自拥有四个子分区 P11、P12、P13 和 P14。如下表所示：

表 3.6.2

<table>
<tr><th>父表</th><th>一级分区</th><th>二级分区</th></tr>
<tr><td rowspan="10">SALES</td><td rowspan="2">P1</td><td>P11_1</td></tr>
<tr><td>P11_2</td></tr>
<tr><td rowspan="4">P2</td><td>P11</td></tr>
<tr><td>P12</td></tr>
<tr><td>P13</td></tr>
<tr><td>P14</td></tr>
<tr><td rowspan="4">P3</td><td>P11</td></tr>
<tr><td>P12</td></tr>
<tr><td>P13</td></tr>
<tr><td>P14</td></tr>
</table>

例 7 创建一个三级分区，更多级别的分区表的建表语句按照语法类推。

```
CREATE TABLE STUDENT(
NAME VARCHAR(20),
AGE INT,
SEX VARCHAR(10) CHECK (SEX IN ('MAIL','FEMAIL')),
GRADE INT CHECK (GRADE IN (7,8,9))
)
PARTITION BY LIST(GRADE)
  SUBPARTITION BY LIST(SEX) SUBPARTITION TEMPLATE
  (
     SUBPARTITION Q1 VALUES('MAIL'),
     SUBPARTITION Q2 VALUES('FEMAIL')
  ),
  SUBPARTITION BY RANGE(AGE) SUBPARTITION TEMPLATE
  (
     SUBPARTITION R1 VALUES LESS THAN (12),
     SUBPARTITION R2 VALUES LESS THAN (15),
     SUBPARTITION R3 VALUES LESS THAN (MAXVALUE)
   )
(
  PARTITION P1 VALUES (7),
```

```
  PARTITION P2 VALUES (8),
  PARTITION P3 VALUES (9)
);
```

本例子中各分区表的表名详细介绍如下：

表 3.6.3

父表	一级分区	二级分区	三级分区
STUDENT	P1	Q1	R1
			R2
			R3
		Q2	R1
			R2
			R3
	P2	Q1	R1
			R2
			R3
		Q2	R1
			R2
			R3
	P3	Q1	R1
			R2
			R3
		Q2	R1
			R2
			R3

3.6.2 表修改语句

3.6.2.1 修改数据库表

为了满足用户在建立应用系统的过程中需要调整数据库结构的要求，DM 系统提供表修改语句。可对表的结构进行全面的修改，包括修改表名、列名、增加列、删除列、修改列类型、增加表级约束、删除表级约束、设置列缺省值、设置触发器状态等一系列修改。系统只提供外部表的文件（控制文件或数据文件）路径修改功能，如果想更改外部表的表结构，可以通过重建外部表来实现。

语法格式

```
ALTER TABLE [<模式名>.]<表名> <修改表定义子句>
<修改表定义子句> ::=
MODIFY <列定义>|
ADD [COLUMN] <列定义>|
ADD [COLUMN] (<列定义> {,<列定义>})|
REBUILD COLUMNS|
DROP [COLUMN] <列名> [RESTRICT | CASCADE] |
ADD [CONSTRAINT [<约束名>] ] <表级约束子句> [<CHECK 选项>] [<失效生效选项>]|
```

```
DROP CONSTRAINT <约束名> [RESTRICT | CASCADE] |
ALTER [COLUMN] <列名> SET DEFAULT <列缺省值表达式>|
ALTER [COLUMN] <列名> DROP DEFAULT |
ALTER [COLUMN] <列名> RENAME TO <列名> |
ALTER [COLUMN] <列名> SET <NULL | NOT NULL>|
ALTER [COLUMN] <列名> SET [NOT] VISIBLE|
RENAME TO <表名> |
ENABLE ALL TRIGGERS |
DISABLE ALL TRIGGERS |
MODIFY <空间限制子句>|
MODIFY CONSTRAINT <约束名> TO <表级约束子句> [<CHECK 选项>] [RESTRICT | CASCADE]|
MODIFY CONSTRAINT <约束名> ENABLE [<CHECK 选项>]|
MODIFY CONSTRAINT <约束名> DISABLE [RESTRICT | CASCADE] |
WITH COUNTER |
WITHOUT COUNTER |
MODIFY PATH <外部表文件路径> |
DROP IDENTITY|
ADD [COLUMN] <列名>[<IDENTITY 子句>]|
ENABLE CONSTRAINT <约束名> [<CHECK 选项>]|
DISABLE CONSTRAINT <约束名> [RESTRICT | CASCADE] |
DEFAULT DIRECTORY <目录名>|
LOCATION ('<文件名>')|
ENABLE USING LONG ROW|
ADD LOGIC LOG |
DROP LOGIC LOG |
WITHOUT ADVANCED LOG |
TRUNCATE ADVANCED LOG
<空间限制子句> ::=请参考 3.6.1.1 定义数据库基表
<表级约束子句> ::=请参考 3.6.1.1 定义数据库基表
<列定义><IDENTITY 子句>::=请参考 3.6.1.1 定义数据库基表
<CHECK 选项>::=[NOT]CHECK
```

参数

1． <模式名> 指明被操作的基表属于哪个模式，缺省为当前模式；

2． <表名> 指明被操作的基表的名称；

3． <列名> 指明修改、增加或被删除列的名称；

4． <数据类型> 指明修改或新增列的数据类型；

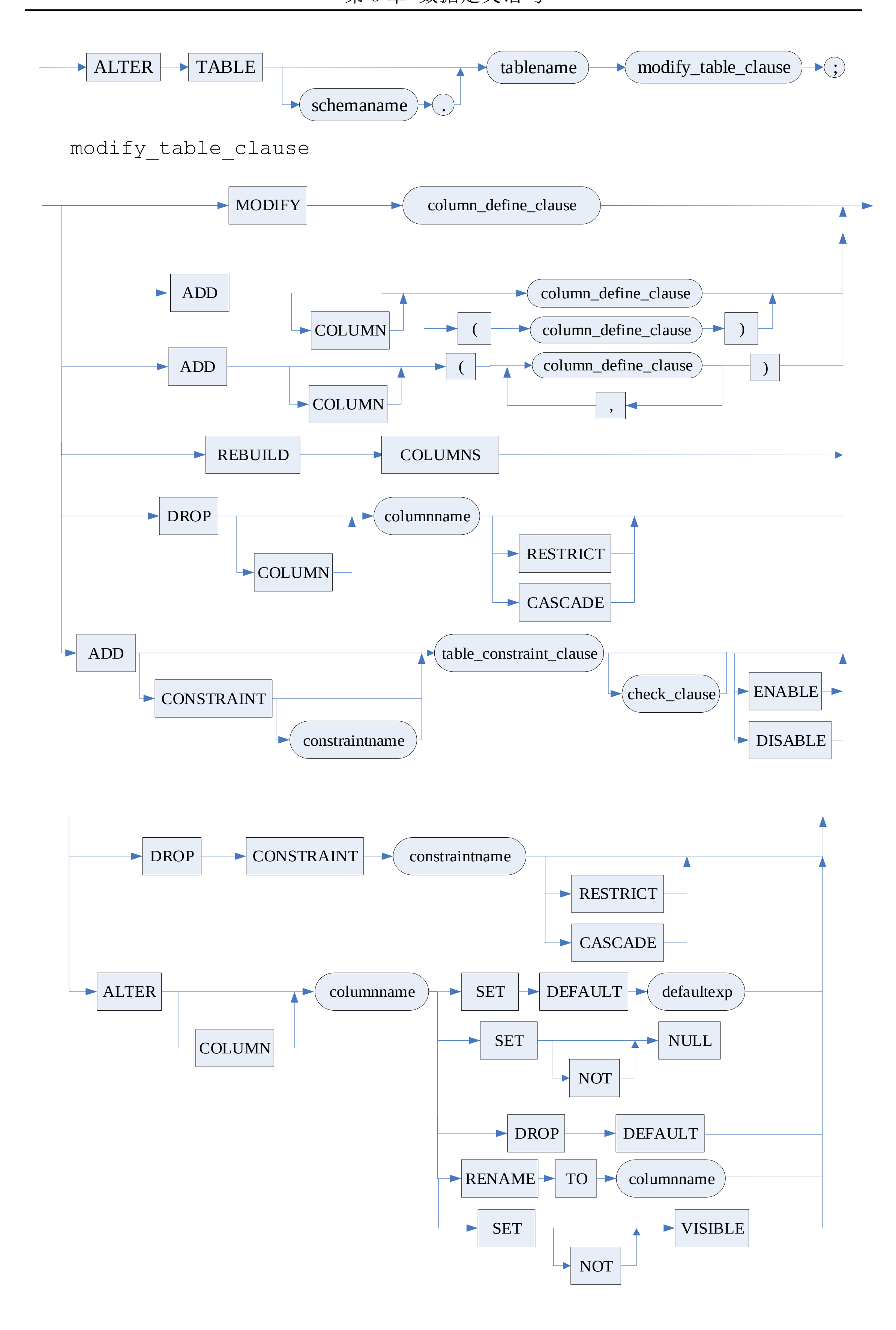
5． <列缺省值> 指明新增/修改列的缺省值，其数据类型与新增/修改列的数据类型一致；

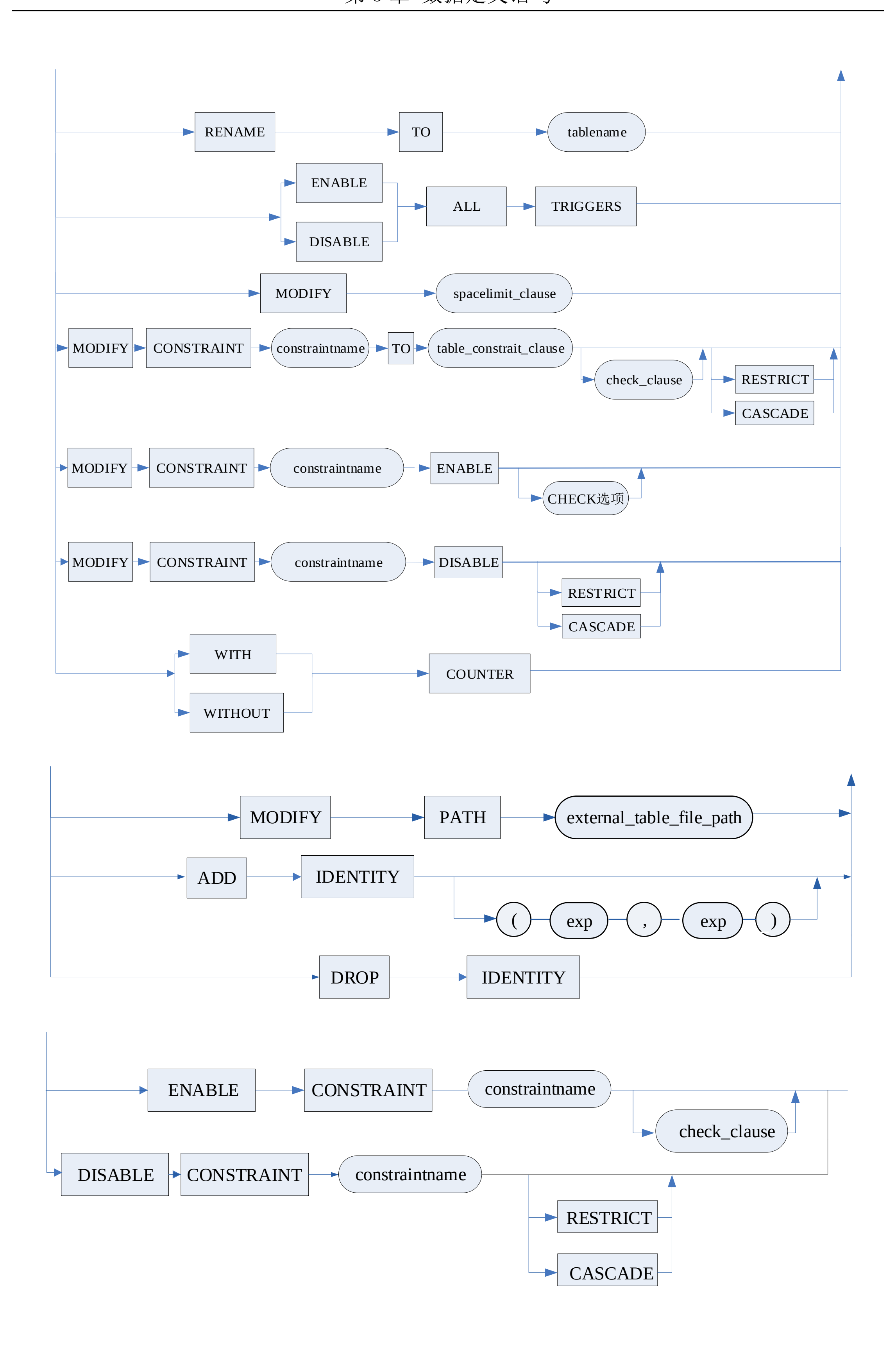
6． <空间限制子句> 分区表不支持修改空间限制；

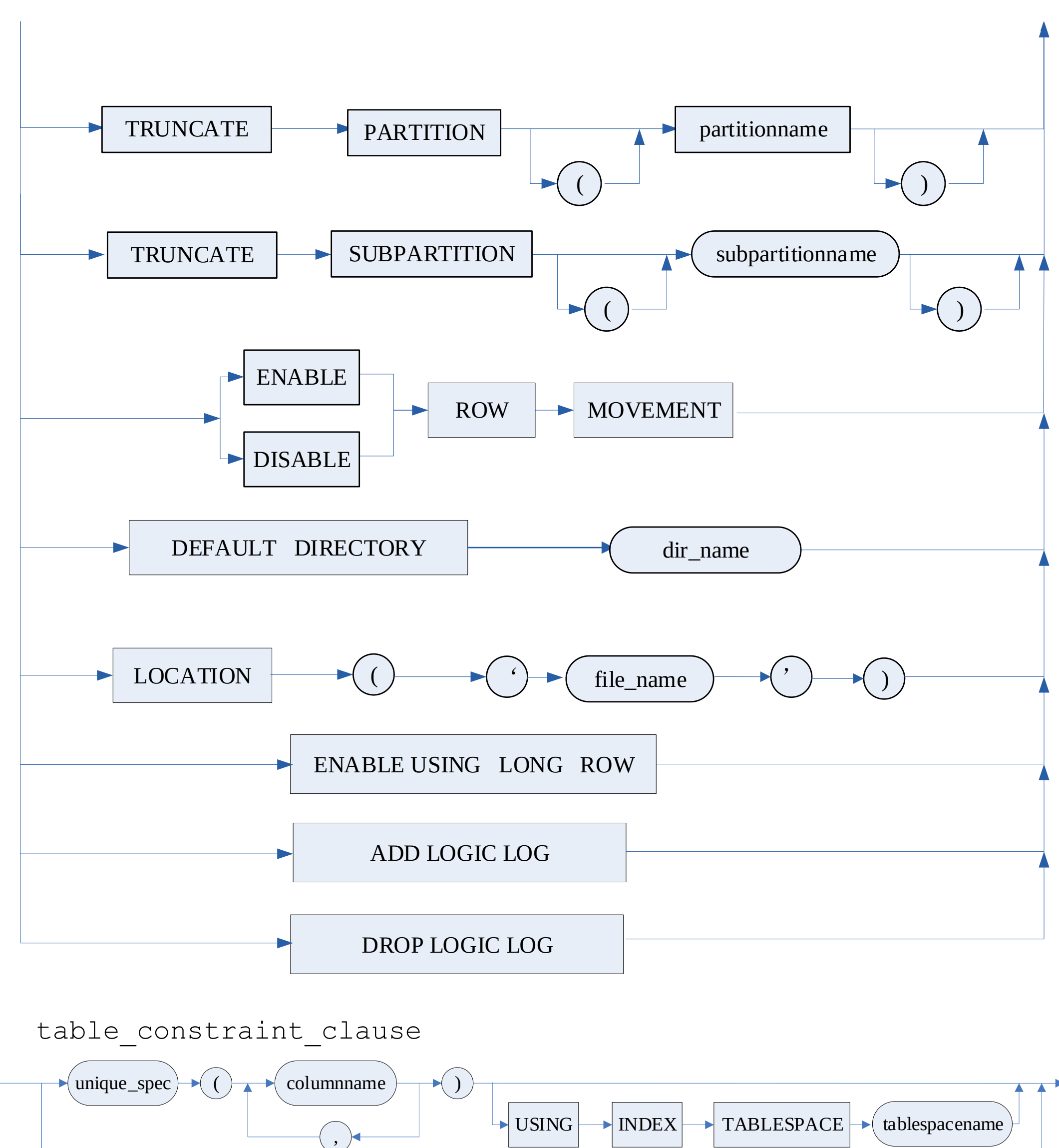
7． <CHECK 选项> 设置在添加外键约束的时候，是否对表中的数据进行约束检查；在添加约束、修改约束和使约束生效时，不指明 CHECK 属性，默认 CHECK；

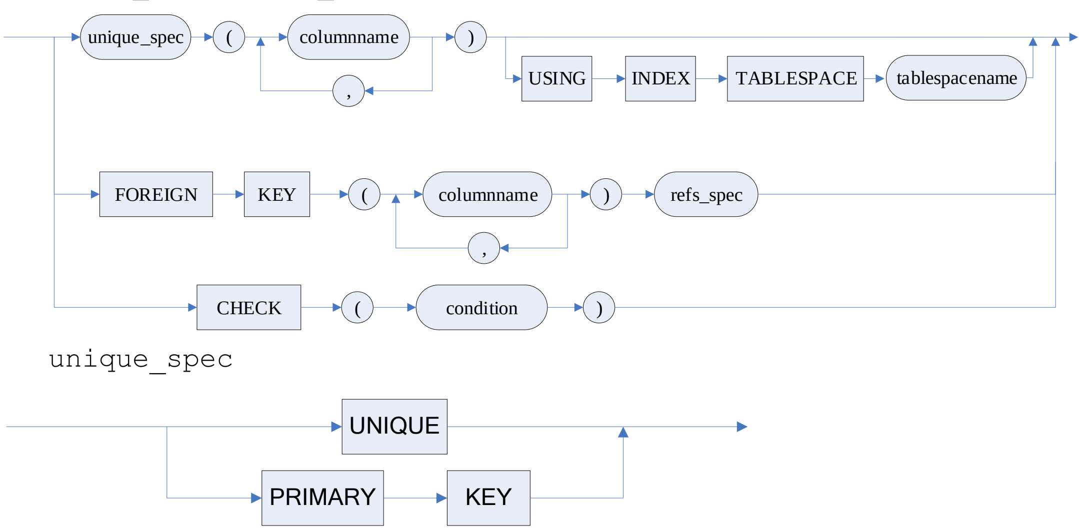
8． <外部表文件路径>指明新的文件在操作系统下的路径＋新文件名。数据文件的存放路径符合 DM 安装路径的规则，且该路径必须是已经存在的。

图例


ALTER
TABLE
schemaname
.
tablename
modify_table_clause
;
modify_table_clause
MODIFY
column_define_clause
ADD
COLUMN
column_define_clause
(
column_define_clause
)
ADD
COLUMN
(
column_define_clause
)
,
REBUILD
COLUMNS
DROP
COLUMN
columnname
RESTRICT
CASCADE
ADD
CONSTRAINT
constraintname
table_constraint_clause
check_clause
ENABLE
DISABLE
DROP
CONSTRAINT
constraintname
RESTRICT
CASCADE
ALTER
COLUMN
columnname
SET
DEFAULT
defaultexp
SET
NOT
NULL
DROP
DEFAULT
RENAME
TO
columnname
SET
NOT
VISIBLE


RENAME
TO
tablename
ENABLE
DISABLE
ALL
TRIGGERS
MODIFY
spacelimit_clause
MODIFY
CONSTRAINT
constraintname
TO
table_constrait_clause
check_clause
RESTRICT
CASCADE
MODIFY
CONSTRAINT
constraintname
ENABLE
CHECK选项
MODIFY
CONSTRAINT
constraintname
DISABLE
RESTRICT
CASCADE
WITH
WITHOUT
COUNTER
MODIFY
PATH
external_table_file_path
ADD
IDENTITY
(
exp
,
exp
)
DROP
IDENTITY
ENABLE
CONSTRAINT
constraintname
check_clause
DISABLE
CONSTRAINT
constraintname
RESTRICT
CASCADE





语句功能

供拥有 DBA 权限的用户或该表的建表者或具有 ALTER ANY TABLE 权限的用户对表的定义进行修改，修改包括：

1. 修改一列的数据类型、精度、刻度，设置列上的 DEFAULT、NOT NULL、NULL；
2. 增加一列及该列上的列级约束；
3. 重建表上的聚集索引数据，消除附加列；

4. 删除一列；
5. 增加、删除表上的约束；
6. 启用、禁用表上的约束；
7. 表名/列名的重命名；
8. 修改触发器状态；
9. 修改表的最大存储空间限制；
10. 修改外部表的文件路径；
11. 修改超长记录（变长字符串）存储方式；
12. 增加、删除表上记录物理逻辑日志的功能。

使用说明

1. 具有 DBA 权限的用户或该表的建表者才能执行此操作；
2. 各个子句中都可以含有单个或多个列定义（约束定义），单个列定义（约束定义）和多个列定义（约束定义）应该在它们的定义外加一层括号，括号要配对；
3. MODIFY COLUMN 说明：
   1) 使用 MODIFY COLUMN 时，支持不指定<数据类型>语法。在这种格式下，不支持<STORAGE 子句>和<存储加密子句>属性的修改。仅修改<列定义子句>中指定的属性，其它属性保留原状。
   2) 使用 MODIFY COLUMN 时，不能更改聚集索引的列或者函数索引的列，位图、位图连接索引的列以及自增列不允许被修改。
   3) 修改引用约束中引用和被引用的列时，列类型与引用的列类型或列类型与被引用的列类型须兼容。
   4) 使用 MODIFY COLUMN 子句能修改的约束有列上的 NULL/NOT NULL 约束、CHECK 约束、 唯一约束、 主键约束（不包括聚集主键约束）；如果修改数据类型不变，则列上现有值必须满足约束条件；不允许被修改为自增列。
   5) 使用 MODIFY COLUMN 子句添加 NULL 约束时，不能重复添加；如果某列现有的值均非空，则允许添加 NOT NULL；主键约束和 NULL 约束不允许同时添加；表上已经有主键约束，则不可以添加主键约束。列上已有唯一约束和主键约束，则不允许添加唯一约束和主键约束。添加了唯一/主键约束就不能添加唯一/主键约束。
   6) 使用 MODIFY COLUMN 修改可更改列的数据类型时，若该表中无元组，则可任意修改其数据类型、长度、精度或量度和加密属性；若表中有元组，则系统会尝试修改其数据类型、长度、精度或量度，如果修改不成功，则会报错返回。
   7) 无论表中有无元组，多媒体数据类型和非多媒体数据类型都不能相互转换，CLOB 类型与 BLOB 类型也不能相互转换；BFILE 类型与其他所有类型都不能相互转换。
   8) 使用 MODIFY COLUMN 不能修改类型为多媒体数据类型的列。
   9) 修改有默认值的列的数据类型时，原数据类型与新数据类型必须是可以转换的，否则即使数据类型修改成功，但在进行插入等其他操作时，仍会出现数据类型转换错误。
4. ADD COLUMN 说明：
   1) 当设置 INI 参数 ALTER_TABLE_OPT 为 1 时，添加列采用查询插入实现，可能会导致 ROWID 的改变；ALTER_TABLE_OPT 为 2 时，系统开启快速加列功能，对于没有默认值或者默认值为 NULL 的新列，系统内部会标记为附加列，能够达到瞬间加列的效果，此时 ROWID 不会改变，若有默认值且默认值不为

NULL，则仍旧采取查询插入实现。

2） 使用 ADD COLUMN 时，新增列名之间、新增列名与该基表中的其它列名之间均不能重复。若新增列有缺省值，则已存在的行的新增列值是其缺省值。添加新列对于任何涉及表的约束定义没有影响，对于涉及表的视图定义不会自动增加。例如：如果用"*"为一个表创建一个视图，那么后加入的新列不会加入到该视图。

3） 使用 ADD COLUMN 时，还有以下限制条件：

a） 列定义中如果带有列约束，只能是对该新增列的约束，但不支持引用约束；列级约束可以带有约束名，系统中同一模式下的约束名不得重复，如果不带约束名，系统自动为此约束命名；

b） 如果表上没有元组，列可以指定为 NOT NULL；如果表中有元组，对于已有列可以指定同时有 DEFAULT 和 NOT NULL，新增列不能指定 NOT NULL；

c） 该列可指定为 CHECK；

d） 该列可指定为 FOREIGN KEY；

e） 允许向空数据的表中，添加自增列；

f） 不支持对列存储表加列；

g） MPP 环境下不支持 ADD [COLUMN] <列名>[<IDENTITY 子句>]。

5． 使用 REBUILD COLUMNS 对添加过新列的表，重建表中的索引数据，消除附加列。水平分区表增加列后不允许使用 REBUILD COLUMNS；

6． 用 DROP COLUMN 子句删除一列有两种方式：RESTRICT 和 CASCADE。RESTRICT 方式为缺省选项，确保只有不被其他对象引用的列才被删除。无论哪种方式，表中的唯一列不能被删除。RESTRICT 方式下，当设置 INI 参数 DROP_CASCADE_VIEW = 1 或者 COMPATIBLE_MODE = 1 时，下列类型的列不能被删除：被引用列、建有视图的列、有 CHECK 约束的列。删除列的同时将删除该列上的约束。CASCADE 方式下，将删除这一列上的引用信息和被引用信息、引用该列的视图、索引和约束；系统允许直接删除 PK 列。但被删除列为 CLUSTER PRIMARY KEY 类型时除外，此时不允许删除；

7． ADD CONSTRAINT 子句用于添加表级约束。表级约束包括：PRIMARY KEY 约束（不包括聚集主键约束）、UNIQUE 约束、引用约束（REFERENCES）和检查约束（CHECK）。添加表级约束时可以带有约束名，系统中同一模式下的约束名不得重复，如果不带约束名，系统自动为此约束命名；

用 ADD CONSTRAINT 子句添加约束时，对于该基表上现有的全部元组要进行约束违规验证：

1） 添加一个 PRIMARY KEY 约束时，要求将成为关键字的字段上无重复值且值非空，并且表上没有定义主关键字；

2） 添加一个 UNIQUE 约束时，要求将成为唯一性约束的字段上不存在重复值，但允许有空值；

3） 添加一个 REFERENCES 约束时，要求将成为引用约束的字段上的值满足该引用约束。

4） 添加一个 CHECK 约束或外键时，要求该基表中全部的元组满足该约束。

8． DROP CONSTRAINT 子句用于删除表级约束，表级约束包括：主键约束（PRIMARY KEY）、唯一性约束（UNIQUE）、引用约束（REFERENCES）和检查约束（CHECK）。用 DROP CONSTRAINT 子句删除一约束时，同样有 RESTRICT 和 CASCADE 两种方式。当删除主键或唯一性约束时，系统自动创建的索引也将一起删除。如果打算删除一个主键约束或一个唯

一性约束而它有外部约束，除非指定 CASCADE 选项，否则将不允许删除。也就是说，指定 CASCADE 时，删除的不仅仅是用户命名的约束，还有任何引用它的外部约束；

9. 空间限制子句说明：修改表的最大存储空间限制时，空间大小以 M 为单位，取值范围在表的已占用空间和 1024*1024 之间，还可以利用 UNLIMITED 关键字去掉表的空间限制；

10. MODIFY PATH 说明：

1) 只能用于修改外部表的文件路径，文件包括控制文件和数据文件；
2) 只支持对应创建方式的文件路径修改：原表使用控制文件创建，则只支持修改控制文件路径；原表使用数据文件创建，则只支持修改数据文件路径。

11. DEFAULT DIRECTORY <目录名>和 LOCATION ('<文件名>')，专门用于修改外部表路径名和文件名；

12. ADD/DROP LOGIC LOG 开启或关闭物理逻辑日志记录；

13. WITHOUT/ TRUNCATE ADVANCED LOG 用于删除日志辅助表和清除日志辅助表数据，具体可参考 19.3.1 节。

举例说明

例1 产品的评论表中COMMENTS、PRODUCT_REVIEWID、PRODUCTID、RATING列都不允许修改，分别因为：COMMENTS为多媒体数据类型；PRODUCT_REVIEWID上定义有关键字，属于用于索引的列；PRODUCTID用于引用约束（包括引用列和被引用列）；RATING用于CHECK约束。另外，关联有缺省值的列也不能修改。而其他列都允许修改。假定用户为SYSDBA，如将评论人姓名的数据类型改为VARCHAR(8)，并指定该列为NOT NULL，且缺省值为'刘青'。

```
ALTER TABLE PRODUCTION.PRODUCT_REVIEW MODIFY NAME VARCHAR(8) DEFAULT '刘青' NOT NULL;
```

此语句只有在表中无元组的情况下才能成功。

例2 具有DBA权限的用户需要对EMPLOYEE_ADDRESS表增加一列，列名为ID（序号），数据类型为INT，值小于10000。

```
ALTER TABLE RESOURCES.EMPLOYEE_ADDRESS ADD ID INT PRIMARY KEY CHECK (ID <10000);
```

如果该表上没有元组，且没有PRIMARY KEY，则可以将新增列指定为 PRIMARY KEY。表上没有元组时也可以将新增列指定为UNIQUE，但同一列上不能同时指定PRIMARY KEY和UNIQUE两种约束。

例3 具有DBA权限的用户需要对ADDRESS表增加一列，列名为PERSONID，数据类型为INT，定义该列为DEFAULT和NOT NULL。

```
ALTER TABLE PERSON.ADDRESS ADD PERSONID INT DEFAULT 10 NOT NULL;
```

如果表上没有元组，新增列可以指定为 NOT NULL；如果表上有元组且都不为空，该列可以指定同时有 DEFAULT 和 NOT NULL，不能单独指定为 NOT NULL。

例 4 具有 DBA 权限的用户需要删除 PRODUCT 表的 PRODUCT_SUBCATEGORYID 一列。

```
ALTER TABLE PRODUCTION.PRODUCT DROP PRODUCT_SUBCATEGORYID CASCADE;
```

删除PRODUCT_SUBCATEGORYID这一列必须采用CASCADE方式，因为该列引用了PRODUCT_SUBCATEGORY表的PRODUCT_SUBCATEGORYID。

例5 具有DBA权限的用户需要在PRODUCT表上增加UNIQUE约束，UNIQUE字段为NAME。

```
ALTER TABLE PRODUCTION.PRODUCT ADD CONSTRAINT CONS_PRODUCTNAME UNIQUE(NAME);
```

用ADD CONSTRAINT子句添加约束时，对于该基表上现有的全部元组要进行约束违规

验证。在这里，分为三种情况：

1. 如果表商场登记里没有元组，则上述语句一定执行成功；
2. 如果表商场登记里有元组，并且欲成为唯一性约束的字段商场名上不存在重复值，则上述语句执行成功；
3. 如果表商场登记里有元组，并且欲成为唯一性约束的字段商场名上存在重复值，则上述语句执行不成功，系统报错“无法建立唯一性索引”。

如果语句执行成功，用户通过

```
CALL SP_TABLEDEF('PRODUCTION', 'PRODUCT');
```

可以看到，修改后的商场登记的表结构显示为：

```
CREATE TABLE "PRODUCTION"."PRODUCT"
(
"PRODUCTID" INTEGER IDENTITY(1, 1) NOT NULL,
"NAME" VARCHAR(50) NOT NULL,
"AUTHOR" VARCHAR(25) NOT NULL,
"PUBLISHER" VARCHAR(50) NOT NULL,
"PUBLISHTIME" DATE NOT NULL,
"PRODUCT_SUBCATEGORYID" INTEGER NOT NULL,
"PRODUCTNO" VARCHAR(25) NOT NULL,
"SATETYSTOCKLEVEL" SMALLINT NOT NULL,
"ORIGINALPRICE" DEC(19,4) NOT NULL,
"NOWPRICE" DEC(19,4) NOT NULL,
"DISCOUNT" DEC(2,1) NOT NULL,
"DESCRIPTION" CLOB,
"PHOTO" BLOB,
"TYPE" VARCHAR(5),
"PAPERTOTAL" INTEGER,
"WORDTOTAL" INTEGER,
"SELLSTARTTIME" DATE NOT NULL,
"SELLENDTIME" DATE,
PRIMARY KEY("PRODUCTID"),
UNIQUE("PRODUCTNO"),
CONSTRAINT "CONS_PRODUCTNAME" UNIQUE("NAME"),
FOREIGN KEY("PRODUCT_SUBCATEGORYID") REFERENCES
"PRODUCTION"."PRODUCT_SUBCATEGORY"("PRODUCT_SUBCATEGORYID")) STORAGE(ON "MAIN",
CLUSTERBTR) ;
```

例6 假定具有DBA权限的用户需要删除PRODUCT表上的NAME列的UNIQUE约束。当前的PRODUCT表结构请参见例5。

删除表约束，首先需要得到该约束对应的约束名，用户可以查询系统表 SYSCONS，如下所示。

```
SELECT NAME FROM SYS.SYSOBJECTS WHERE SUBTYPE$='CONS' AND PID =
(SELECT ID FROM SYS.SYSOBJECTS WHERE NAME = 'PRODUCT' AND TYPE$='SCHOBJ' AND
SCHID=( SELECT ID FROM SYS.SYSOBJECTS  WHERE NAME='PRODUCTION' AND TYPE$='SCH'));
```

该系统表显示商场登记表上的所有PRIMARY KEY、UNIQUE、CHECK约束。查询得到NAME列上UNIQUE约束对应的约束名，这里为CONS_PRODUCTNAME。

然后，可采用以下的语句删除指定约束名的约束。

```
ALTER TABLE PRODUCTION.PRODUCT DROP CONSTRAINT CONS_PRODUCTNAME;
```

语句执行成功。

例 7 建立普通表查看 SELECT COUNT(*) 执行计划，再 ALTER 该表为 WITHOUT COUNTER 属性，再查看执行计划：

```
SQL>CREATE TABLE test(c1 INT);
操作已执行
已用时间:55.811(毫秒). 执行号:2.

SQL>EXPLAIN SELECT COUNT(*) FROM test;

1   #NSET2: [0, 1, 0]
2     #PRJT2: [0, 1, 0]; exp_num(1), is_atom(FALSE)
3       #FAGR2: [0, 1, 0]; sfun_num(1)

已用时间:7.304(毫秒). 执行号:0.

SQL>ALTER TABLE test WITHOUT COUNTER;
操作已执行
已用时间:97.306(毫秒). 执行号:3.

SQL>EXPLAIN SELECT COUNT(*) FROM test;

1   #NSET2: [0, 1, 0]
2     #PRJT2: [0, 1, 0]; exp_num(1), is_atom(FALSE)
3       #AAGR2: [0, 1, 0]; grp_num(0), sfun_num(1)
4         #CSCN2: [0, 1, 0]; INDEX33555472(TEST)

已用时间:5.734(毫秒). 执行号:0.
```

例 8 创建、启用、禁用、删除 PERSON 表约束 unq。

```
ALTER TABLE PERSON.PERSON ADD CONSTRAINT unq UNIQUE(PHONE);
ALTER TABLE PERSON.PERSON ENABLE CONSTRAINT unq;
ALTER TABLE PERSON.PERSON DISABLE CONSTRAINT unq;
ALTER TABLE PERSON.PERSON DROP CONSTRAINT unq;
```

3.6.2.2 修改水平分区表

本节专门列出水平分区表在分区方面的修改。其他方面的常规修改和普通表一样，普通表的修改方法在水平分区表上完全适用，详细请参考3.6.2.1 修改数据库表部分。

语法格式

```
ALTER TABLE [<模式名>.]<表名> <修改表定义子句>
<修改表定义子句> ::=
MODIFY <增加多级分区子表>|
<删除多级分区子表>|
MODIFY <修改 LIST 分区子表>|
```

```
    ADD <水平分区项>|
    DROP PARTITION <分区名>|
    EXCHANGE <PARTITION| SUBPARTITION > <分区名> WITH TABLE [<模式名.>]<表名>|
    SPLIT PARTITION <分区名> AT (<表达式>{,<表达式>}) INTO ({PARTITION <分区名>
<STORAGE 子句>}, {PARTITION <分区名> <STORAGE 子句>})|
    MERGE PARTITIONS <分区编号>,<分区编号>  INTO PARTITION <分区名>|
    MERGE PARTITIONS <分区名>,<分区名>  INTO PARTITION <分区名>|
    SET SUBPARTITION TEMPLATE <分区模板描述项> |
    TRUNCATE PARTITION [(]<分区名>[)] |
    TRUNCATE SUBPARTITION [(]<子分区名>[)]|
    ENABLE ROW MOVEMENT |
    DISABLE ROW MOVEMENT |
    RENAME <修改分区子表名>
    <增加多级分区子表>:: = <PARTITION | SUBPARTITION> <分区名> ADD <<RANGE 子分区
项>|<LIST 子分区项>>
    <删除多级分区子表>:: =DROP SUBPARTITION <分区名> |
                          DROP SUBPARTITION FOR(<分区列值>)
    <修改 LIST 分区子表>:: = <PARTITION | SUBPARTITION> <分区名> <ADD|DROP> VALUES(分
区值[,分区值])
    <修改分区子表名>:: =<修改一级分区子表名>|
                        <修改多级分区子表名>
    <修改一级分区子表名>:: =PARTITION <分区名> TO <新名称>
    <修改多级分区子表名>:: =SUBPARTITION <分区名> TO <新名称>

    <RANGE 子分区项>请参考 3.6.1.4 定义水平分区表
    <HASH 子分区项>请参考 3.6.1.4 定义水平分区表
    <LIST 子分区项>请参考 3.6.1.4 定义水平分区表
    <分区列值>::= <常量|计算表达式{,常量|计算表达式}>

    <水平分区项>::=<RANGE 分区项>|<HASH 分区项>|<LIST 分区项>
    <RANGE 分区项>请参考 3.6.1.4 定义水平分区表
    <HASH 分区项> 请参考 3.6.1.4 定义水平分区表
    <LIST 分区项> 请参考 3.6.1.4 定义水平分区表
    <STORAGE 子句>请参考 3.6.1.1 定义数据库基表

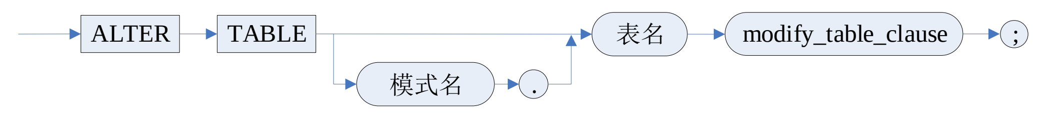
    <分区模板描述项> ::= ([<分区模板描述项 1>])| <分区模板描述项 2>
    <分区模板描述项 1> ::= <RANGE 子分区项> {,<RANGE 子分区项>} |
                          <HASH 子分区项> {,<HASH 子分区项>} |
                          <LIST 子分区项> {,<LIST 子分区项>}
    <分区模板描述项 2> ::= [SUBPARTITIONS] <子分区数> <STORAGE HASH 子句>
    <STORAGE HASH 子句>请参考 3.6.1.4 定义水平分区表
```

参数

1. <模式名> 指明被操作的分区表属于哪个模式，缺省为当前模式；
2. <表名> 指明被操作的分区表的名称；

3． <分区编号> 从 1 开始，2、3、4……以此类推，编号最大值为：实际分区数。

图例



modify_table_clause

MODIFY PARTITION | SUBPARTITION 分区名 ADD RANGE子分区项 | LIST子分区项

DROP SUBPARTITION 分区名 | FOR 分区列值 ,

MODIFY PARTITION | SUBPARTITION 修改LIST分区子表

ADD hash_partition | horizon_partition | list_partition

DROP PARTITION 分区名

exchange_clause

split_clause

merge_clause

SET SUBPARTITION TEMPLATE 分区模板描述项

TRUNCATE PARTITION (分区名)

TRUNCATE SUBPARTITION (子分区名)

ENABLE | DISABLE ROW MOVEMENT

RENAME PARTITION | SUBPARTITION 分区名 TO 新名称



语句功能

供拥有 DBA 权限的用户或该表的建表者或具有 ALTER ANY TABLE 权限的用户对水平分区表的定义进行修改，修改包括：

1． 增加、删除多级分区子表；
2． 修改一级 LIST 分区子表；
3． 拆分、合并、交换水平分区；
4． 修改水平分区子表名称。

使用说明

1． 具有 DBA 权限的用户或该表的建表者才能执行此操作；

2． <增加多级分区子表>说明：

   1） 仅支持 LIST/RANGE 分区的修改；
   2） 不支持添加 HASH 子分区，从父表到待修改的子表之间，不允许存在 HASH 分区；
   3） 新增的分区名不能与子表重名；
   4） 只能增加一个下层子表；
   5） 如果新增子表为中间层子表，可以为其自定义子表或根据模板进行创建；
   6） 禁止对间隔分区新增子表；
   7） 子表名称长度不能超过 128，否则报错。

3． <删除多级分区子表>说明：

   1） 提供两种方式删除二级或二级以上的子表：一是指定子表表名方式；二是指定分区列值的方式；
   2） 删除一级子表仅支持指定子表表名方式；
   3） 待删除子表为唯一分区时，禁止删除；
   4） 采用指定分区列值方式删除子表时，分区列的数目为子表上所有分区列个数总和。例如：一级分区列(c1,c2)； 二级分区列(c3)；三级分区列(c4)，那么在删除二级子表时候，分区列的数目应该为 3 个，三级分区列的数目为 4 个；
   5） 禁止删除 HASH 分区的子表。从父表到待删除子表之间也不允许存在 HASH 分区。

4． <修改 LIST 分区子表>说明：

   1） 支持修改一级、多级分区的 LIST 分区子表范围值。修改后的子分区，通过 sp_tabledef()显示完整的分区定义；
   2） 新增的分区值不能重复；
   3） 不能删除分区所有范围值；
   4） 修改的范围值必须为常量；
   5） 如果表中存在某个范围值的记录，则不能删除此范围值；
   6） 不能重复删除同一个范围值；
   7） 不能增加、删除 default 值；
   8） 不能修改 DEFAULT 分区。

5． MERGE 合并分区，将相邻的两个范围分区合并为一个分区。合并分区通过指定分区号进行，相邻的两个分区的表空间没有特殊要求，合并分区后，被合并分区（分区号小的分区）的数据加入合并分区（分区号大的分区）；

6． SPLIT 拆分分区，将某一个范围分区拆分为相邻的两个分区。拆分分区时指定的常量表达式值不能是原有的分区表范围值。拆分分区时，可以指定分区表空间，也可以不指定；指定的分区表空间不要求一定与原有分区相同；如果不指定表空间缺省使用拆分前分区的表空间；拆分分区会对分区中的数据从主表的角度进行重定位；

7． 对于堆表分区，不论是拆分还是添加分区，都不能改变分区表空间，各子表必须和原表位于同一表空间；

8． 哈希分区支持重命名、删除约束、设置触发器是否启用的修改操作；

9． 范围分区支持分区合并、拆分、增加、删除、交换、重命名、删除约束、设置触发器是否生效操作；

10．LIST 分区支持分区拆分、增加、删除、交换、重命名、删除约束、设置触发器是否生效操作；

11. 对范围分区增加分区值必须是递增的，即只能在最后一个分区后添加分区。LIST 分区增加分区值不能存在于其他已在分区；

12. 当水平分区数仅剩一个时，不允许进行删除分区；

13. 不允许删除被引用表的子分区；

14. EXCHANGE PARTITION/SUBPARTITION 用于交换一级分区/多级子分区的表。交换一级分区，仅范围分区和 LIST 分区支持交换分区，哈希分区表不支持。

1) 普通表交换分区要求分区表与交换表具有相同的结构（相同的表类型、相同的 BRANCH 选项、相同的列结构、相同的索引、相同的分布方式），且不支持包含全局索引的分区表与普通表进行交换分区操作。分区交换并不会校验数据，如交换表的数据是否符合分区范围等，即不能保证分区交换后的分区上的数据符合分区范围；

2) HUGE 表交换分区要求分区表与交换表具有相同的 STORAGE 参数(SECTION，FILESIZE，WITH/WITHOUT DELTA 和 STAT NONE)；HUGE 分区表交换表分区时，如果表空间包含多个路径，则源表与目标表必须存储在同一个表空间。

15. TRUNCATE PARTITION/SUBPARTITION 用于清除指定一级分区子表/子分区子表的数据，对应一级分区子表/子分区子表结构仍然保留。

16. 修改分区模板说明：

1) 支持模板信息从无到有、从有到无的修改；

2) 模板信息定义需要匹配子表类型，并符合对应分区类型的限制，例如范围分区值递增、列表分区不能存在重复值、范围值不能为 NULL 等；

3) 修改模板信息后，之前创建的子表不受影响，后续新增子表按照新的模板信息进行创建；

4) 仅支持修改第二层子表的模板信息。

17. 多级分区表支持下列修改表操作：新增分区、新增列、删除列、删除表级约束、修改表名、设置与删除列的默认值、设置列 NULL 属性、设置列可见性、设置行迁移属性、启用超长记录、WITH DELTA、新增子分区、删除子分区、修改二级分区模板信息；

18. 水平分区表支持的列修改操作除了多级分区表支持的操作外，还支持：设置触发器生效/失效、修改除分区列以外的列名、修改列属性、增加表级主键约束、删除分区、SPLIT/MERGE 分区和交换分区；

19. <修改分区子表名>支持修改所有类型的分区子表名称。修改的子表名不能与现有子表重名。修改的名称需要符合对象命名规则，例如名称长度限制等。

举例说明

例 1 合并分区表，修改分区表。

```
ALTER TABLE PRODUCTION.PRODUCT_INVENTORY MERGE PARTITIONS P1,P2 INTO PARTITION P5;
```

执行后，分区结构如下：

表 3.6.4

PARTITIONNO	P5	P3	P4
VALUES	QUANTITY<=100	100<QUANTITY<=10000	QUANTITY<99999

例 2 在例 1 合并分区表基础上，重新拆分分区表，修改分区表。

```
ALTER TABLE PRODUCTION.PRODUCT_INVENTORY SPLIT PARTITION P3 AT(666) INTO(PARTITION P6,PARTITION P7);
```

执行后，分区结构如下：

表 3.6.5

PARTITIONNO	P5	P6	P7	P4
VALUES	QUANTITY<=100	100<QUANTITY<=666	666<QUANTITY<=10000	10000< QUANTITY

例 3 增加，删除分区子表。

```
DROP TABLE T1 CASCADE;
CREATE TABLE T1 (C1 INT, C2 INT)
  PARTITION BY LIST (C1)
  SUBPARTITION BY RANGE (C2)  SUBPARTITION TEMPLATE (
    SUBPARTITION SP1 VALUES LESS THAN(10),
    SUBPARTITION SP2 VALUES LESS THAN(20))
  (
    PARTITION FP1 VALUES(1,2,3),
    PARTITION FP2 VALUES(4,5,6)
  );
-- 增加一个二级分区 SP3。增加的二级分区名称为 T1_FP1_SP3。
ALTER TABLE T1 MODIFY PARTITION FP1 ADD SUBPARTITION SP3 VALUES LESS THAN(300) ;
-- 为一级 LIST 分区 FP2 增加一个范围值 7。
ALTER TABLE T1 MODIFY PARTITION FP2 ADD VALUES(7);
-- 删除分区子表。采用指定分区列值的方法和指定子表名的方法定位分区子表。指定第二列值为 2，第三列值为 60，可以定位到 FP1_SP3 子表，等同于直接指定子表名 FP1_SP3。
ALTER TABLE T1 DROP SUBPARTITION FOR(2,60);
等同于
ALTER TABLE T1 DROP SUBPARTITION FP1_SP3;
```

例 4 交换分区子表。

```
CREATE HUGE TABLE PARTITION_T1(C1 INT, C2 INT)PARTITION BY RANGE(C1)(
    PARTITION PAR1 VALUES LESS THAN (2),
    PARTITION PAR2 VALUES LESS THAN (10),
    PARTITION PAR3 VALUES LESS THAN (30)
);

INSERT INTO PARTITION_T1 VALUES (1,1);
INSERT INTO PARTITION_T1 VALUES (9,19);
INSERT INTO PARTITION_T1 VALUES (21,22);
COMMIT;
CREATE HUGE TABLE PARTITION_T2(C1 INT, C2 INT);
INSERT INTO PARTITION_T2 VALUES (100,100);
COMMIT;
--将分区表 PAR1 分区和表 PARTITION_T2 进行交换。
ALTER TABLE PARTITION_T1 EXCHANGE PARTITION PAR1 WITH TABLE PARTITION_T2;
--交换之后，数据发生了互换
SQL>SELECT * FROM PARTITION_T1_PAR1;
行号      C1        C2
---------- ----------- -----------
1         100       100
```

```
SQL> SELECT * FROM PARTITION_T2;
行号       C1          C2
---------- ----------- -----------
1          1           1
```

3.6.2.3 修改 HUGE 表

本节专门列出 HUGE 表的修改操作。

语法格式

```
ALTER TABLE [<模式名>.]<表名><修改表定义子句>
<修改表定义子句> ::=
……
ALTER [COLUMN] <列名> SET STAT NONE
ALTER [COLUMN] (<列名>{,<列名>})SET STAT [NONE]|
WITH DELTA|
SET STAT [NONE | SYNCHRONOUS | ASYNCHRONOUS] [<ON | EXCEPT> (<列名>{,<列名>})]|
REFRESH STAT|
FORCE COLUMN STORAGE
```

省略号“……”部分包含两部分。一部分来自 3.6.2.1 修改数据库表，一部分来自 3.6.2.2 修改水平分区表。

来自 3.6.2.1 修改数据库表的部分如下：

```
ALTER [COLUMN] <列名> SET DEFAULT <列缺省值表达式>|
ALTER [COLUMN] <列名> DROP DEFAULT |
ALTER [COLUMN] <列名> RENAME TO <列名> |
RENAME TO <表名> |
ENABLE CONSTRAINT <约束名> [<CHECK 选项>]|
DISABLE CONSTRAINT <约束名> [RESTRICT | CASCADE]
```

来自 3.6.2.2 修改水平分区表的部分如下：

```
EXCHANGE <PARTITION| SUBPARTITION > <分区名> WITH TABLE [<模式名.>]<表名>|
SPLIT PARTITION <分区名> AT (<表达式>{,<表达式>}) INTO ({PARTITION <分区名>
<STORAGE 子句>}, {PARTITION <分区名> <STORAGE 子句>})|
MERGE PARTITIONS <分区编号>,<分区编号>  INTO PARTITION <分区名>|
MERGE PARTITIONS <分区名>,<分区名>  INTO PARTITION <分区名>|
SET SUBPARTITION TEMPLATE <分区模板描述项> |

<分区模板描述项> ::= ([<分区模板描述项 1>])| <分区模板描述项 2>
<分区模板描述项 1> ::= <RANGE 子分区项> {,<RANGE 子分区项>} |
                      <HASH 子分区项> {,<HASH 子分区项>} |
                      <LIST 子分区项> {,<LIST 子分区项>}
<分区模板描述项 2> ::= [SUBPARTITIONS] <子分区数> <STORAGE HASH 子句>
<STORAGE HASH 子句>请参考 3.6.1.4 定义水平分区表
```

参数

1. <模式名> 指明被操作的基表属于哪个模式，缺省为当前模式；
2. <表名> 指明被操作的基表的名称；
3. <列名> 指明修改、增加或被删除列的名称；

4. ALTER [COLUMN] <列名> SET STAT NONE 修改（关闭）某一个列的统计信息属性。

5. ALTER [COLUMN] (<列名>{,<列名>})SET STAT [NONE] 修改表的某一列或多列的统计开关，STAT 为打开，STAT NONE 为关闭。只有表的统计状态是实时或异步时，支持打开或关闭列的统计开关，打开之后列的统计状态和表的统计状态一致；如果表的统计开关是关闭的，不支持打开或关闭列的统计开关。如下表所示：

表 3.6.6 打开或关闭指定列的统计开关

表统计状态	列状态	操作
实时或异步	STAT	当表的状态是实时或异步时，支持打开列统计，统计状态与表统计状态相同
	STAT NONE	当表的状态是实时或异步时，支持关闭列统计
NONE	STAT	当表的状态是 NONE 时，不支持打开或关闭列统计
	STAT NONE	

6. WITH DELTA 表示创建事务型 HUGE 表；WITHOUT DELTA 表示创建非事务型 HUGE 表；

7. SET STAT [NONE | SYNCHRONOUS | ASYNCHRONOUS] [<ON | EXCEPT>(col_lst)]修改表的统计状态。转换规则见下表：

表 3.6.7 设置表的统计信息

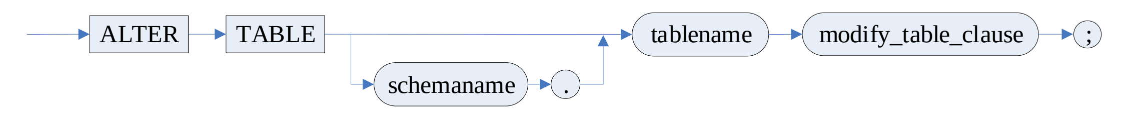
原 STAT	新 STAT	列设置	说明
实时或异步	NONE	不设置	关闭表的统计开关（即所有列都不做统计）
实时	异步	不设置	切换表统计状态
异步	实时		
NONE	实时或异步	必须设置	设置表的统计状态，并打开指定列的统计
前后状态一致		设置或不设置	报错

转换规则注意事项如下：

1) [NONE | SYNCHRONOUS | ASYNCHRONOUS] [ON | EXCEPT (col_lst)] 的详细意义请参考 3.6.1.3 定义 HUGE 表<storage 子句 1>；
2) 如果是从 NONE 到打开统计，必须要带上列设置<ON | EXCEPT>(col_lst)，否则报错；
3) 如果是从打开到 NONE，影响所有列；
4) 如果是实时和异步之间的切换，不支持列设置，不影响各列上的统计状态，只是切换计算统计信息的方式；
5) 如果修改后是实时计算统计信息，对于表中需要计算统计信息的列，要将统计信息的值设置为准确的值。

8. REFRESH STAT 刷新辅助表$AUX 的统计信息，为了更新不准确的数据区的信息；
9. FORCE COLUMN STORAGE 强制把 R 表数据写入数据文件，仅 WITH DELTA 的 HUGE 表支持。

图例



modify_table_clause

ALTER [COLUMN] columnname SET STAT NONE

或

ALTER [COLUMN] (columnname {, columnname}) SET STAT [NONE]

或

WITH DELTA

或

STAT [NONE | SYNCHRONOUS | ASYNCHRONOUS] [ON | EXCEPT (columnname {, columnname})]
REFRESH STAT

或

FORCE COLUMN STORAGE

"……"部分的语法图请参考 3.6.2.1 修改数据库表和 3.6.2.2 修改水平分区表。

举例说明

创建一个 HUGE 表 STUDENT，表的区大小为 65536 行，文件大小为 64M，S_COMMENT 列指定的区大小为不做统计信息，其它列（默认）都做统计信息。

```
CREATE HUGE TABLE STUDENT
(
    S_NO             INT,
    S_CLASS      VARCHAR,
    S_COMMENT            VARCHAR(79)  STORAGE(STAT NONE)
)STORAGE(SECTION(65536), WITH DELTA, FILESIZE(64));
```

重新设置列 S_COMMENT，打开 S_COMMENT 的统计开关，对该列做统计信息。

```
ALTER TABLE STUDENT ALTER COLUMN (S_COMMENT)  SET STAT;
```

修改表 STUDENT 的统计状态为关闭，即所有列都不做统计信息。

```
ALTER TABLE STUDENT SET STAT NONE;
```

修改表 STUDENT 的统计状态为 SYNCHRONOUS，并打开 S_NO、S_CLASS、S_COMMENT 三列的统计开关，打开之后，列的统计状态和表的统计状态一致。

```
ALTER TABLE STUDENT SET STAT SYNCHRONOUS ON (S_NO, S_CLASS, S_COMMENT);
```

3.6.3 基表删除语句

DM 系统允许用户随时从数据库中删除基表。

语法格式

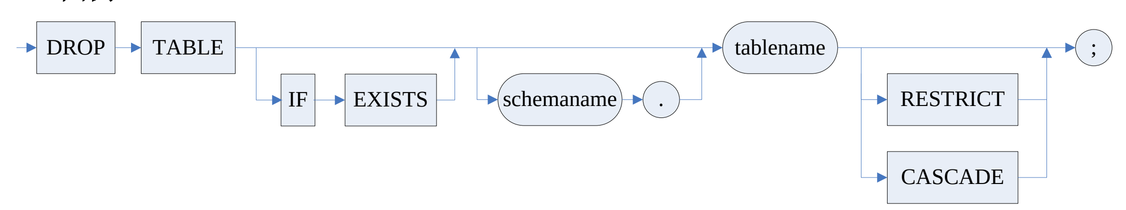
```
DROP TABLE [IF EXISTS] [<模式名>.]<表名> [RESTRICT|CASCADE];
```

参数

1． <模式名> 指明被删除基表所属的模式，缺省为当前模式；

2． <表名> 指明被删除基表的名称。

图例



语句功能

供具有 DBA 权限的用户或该表的拥有者删除基表。

使用说明

1． 删除不存在的基表会报错。若指定IF EXISTS关键字后，删除不存在的表，不会报错；

2． 表删除有两种方式：RESTRICT/CASCADE方式（外部基表除外）。其中RESTRICT为缺省值。如果以CASCADE方式删除该表，将删除表中唯一列上和主关键字上的引用完整性约束，当设置INI参数DROP_CASCADE_VIEW值为1时，还可以删除所有建立在该基表上的视图。如果以RESTRICT方式删除该表，要求该表上已不存在任何视图以及引用完整性约束，否则DM返回错误信息，而不删除该表；

3． 该表删除后，在该表上所建索引也同时被删除；

4． 该表删除后，所有用户在该表上的权限也自动取消，以后系统中再建同名基表是与该表毫无关系的表；

5． 删除外部基表无需指定RESTRICT和CASCADE关键字。

举例说明

例 1 用户 SYSDBA 删除 PERSON 表。

```
DROP TABLE PERSON.PERSON CASCADE;
```

例 2 假设当前用户为用户 SYSDBA。现要删除 PERSON_TYPE 表。为此，必须先删除 VENDOR_PERSON 表，因为它们之间存在着引用关系，VENDOR_PERSON 表为引用表，PERSON_TYPE 表为被引用表。

```
DROP TABLE PURCHASING.VENDOR_PERSON;
DROP TABLE PERSON.PERSON_TYPE;
```

也可以使用 CASCADE 强制删除 PERSON_TYPE 表，但是 VENDOR_PERSON 表仍然存在，只是删除了 PERSON_TYPE 表的引用约束。

```
DROP TABLE PERSON.PERSON_TYPE CASCADE;
```

3.6.4 基表数据删除语句

DM 可以从表中删除所有记录。

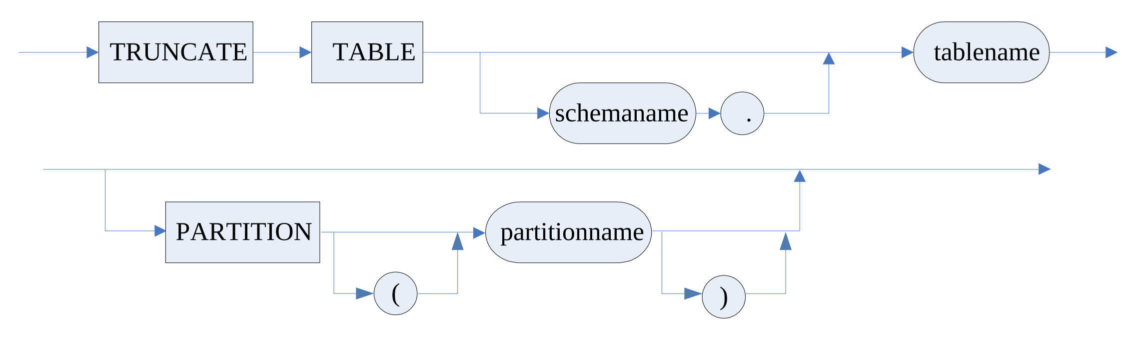
语法格式

```
TRUNCATE TABLE [<模式名>.]<表名>[PARTITION [(]<分区名>[)]];
```

参数

1. <模式名> 指明表所属的模式，缺省为当前模式；
2. <表名> 指明被删除记录的表的名称。
3. <分区名> 指明被删除分区表的名称。

图例



语句功能

供具有 DBA 权限的用户或该表的拥有者从表中删除所有记录。

使用说明

1. TRUNCATE命令一次性删除指定表的所有记录，比用DELETE命令删除记录快。但TRUNCATE不会触发表上的DELETE触发器；
2. 如果TRUNCATE删除的表上有被引用关系，则此语句失败；
3. TRUNCATE不同于DROP命令，因为它保留了表结构及其上的约束和索引信息；
4. TRUNCATE 命令只能用来删除表的所有的记录，而DELETE命令可以只删除表的部分记录。

举例说明

例 假定删除模式 PRODUCTION 中的 PRODUCT_REVIEW 表中所有的记录。

```
TRUNCATE TABLE PRODUCTION.PRODUCT_REVIEW;
```

3.6.5 事务型 HUGE 表数据重整

事务型 HUGE 表删除和更新的数据都存储在对应行辅助表中，查询时还需要扫描行辅助表。当随着时间推移，行辅助表中数据量较大时，需要对数据进行重整，以免影响事务型 HUGE 表的查询效率。数据重整将事务型 HUGE 表中涉及更新和删除操作的数据区进行区内数据重整，重整后 HUGE 表中每个数据区内的数据紧密排列，但数据不会跨区移动。需要注意的是，数据重整会导致 ROWID 发生变化。

仅 WITH DELTA 的 HUGE 表支持数据重整。

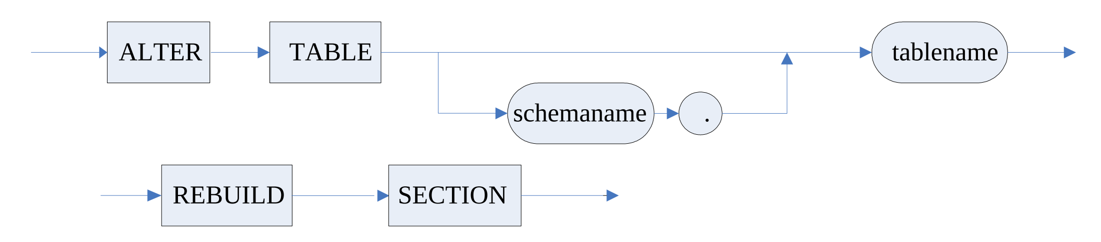
语法格式

```
ALTER TABLE [<模式名>.]<表名> REBUILD SECTION;
```

参数

1. <模式名> 指明表所属的模式，缺省为当前模式；
2. <表名> 指明被删除记录的表的名称；

图例



语句功能

具有 DBA 权限的用户或该表的建表者才能执行此操作。

举例说明

例 对事务型HUGE表ORDERS进行重整。

```
ALTER TABLE ORDERS REBUILD SECTION;
```

3.7 管理索引

3.7.1 索引定义语句

为了提高系统的查询效率，DM 系统提供了索引。但也需要注意，索引会降低那些影响索引列值的命令的执行效率，如 INSERT、UPDATE、DELETE 的性能，因为 DM 不但要维护基表数据还要维护索引数据。

语法格式

```
    CREATE [OR REPLACE] [CLUSTER|NOT PARTIAL][UNIQUE | BITMAP| SPATIAL] INDEX <索引名>
    ON [<模式名>.]<表名>(<索引列定义>{,<索引列定义>}) [GLOBAL] [<STORAGE 子句>] [NOSORT] [ONLINE];
    <索引列定义>::= <索引列表达式>[ASC|DESC]
    <STORAGE 子句>::=<STORAGE 子句 1>|<STORAGE 子句 2>
    <STORAGE 子句 1>::= STORAGE(<STORAGE1 项> {,<STORAGE1 项>})
    <STORAGE1 项> ::=
        [INITIAL <初始簇数目>] |
        [NEXT <下次分配簇数目>] |
        [MINEXTENTS <最小保留簇数目>] |
        [ON <表空间名>] |
        [FILLFACTOR  <填充比例>]|
        [BRANCH  <BRANCH 数>]|
        [BRANCH  (<BRANCH 数>, <NOBRANCH 数>)]|
        [NOBRANCH ]|
        [<CLUSTERBTR>]|
        [SECTION (<区数>)]|
        [STAT NONE]
    <STORAGE 子句 2>::= STORAGE(<STORAGE2 项> {,<STORAGE2 项>})
    <STORAGE2 项> ::=  [ON <表空间名>]|[STAT NONE]
```

参数

1. UNIQUE 指明该索引为唯一索引；

2. BITMAP 指明该索引为位图索引；

3. SPATIAL 指明该索引为空间索引；

4. CLUSTER 指明该索引为聚簇索引（也叫聚集索引），不能应用到函数索引中；

5. NOT PARTIAL 指明该索引为非聚簇索引，缺省即为非聚簇索引；

6. <索引名> 指明被创建索引的名称，索引名称最大长度128字节；

7. <模式名> 指明被创建索引的基表属于哪个模式，缺省为当前模式；

8. <表名> 指明被创建索引的基表的名称；

9. <索引列定义> 指明创建索引的列定义。其中空间索引列的数据类型必须是DMGEO包内的空间类型，如ST_GEOMETRY等；

10.<索引列表达式> 指明被创建的索引列可以为表达式；

11.GLOBAL 指明该索引为全局索引，仅堆表的水平分区表支持该选项，非水平分区表忽略该选项。堆表上的PRIMARY KEY会自动变为全局索引；

12.ASC 递增顺序；

13.DESC 递减顺序；

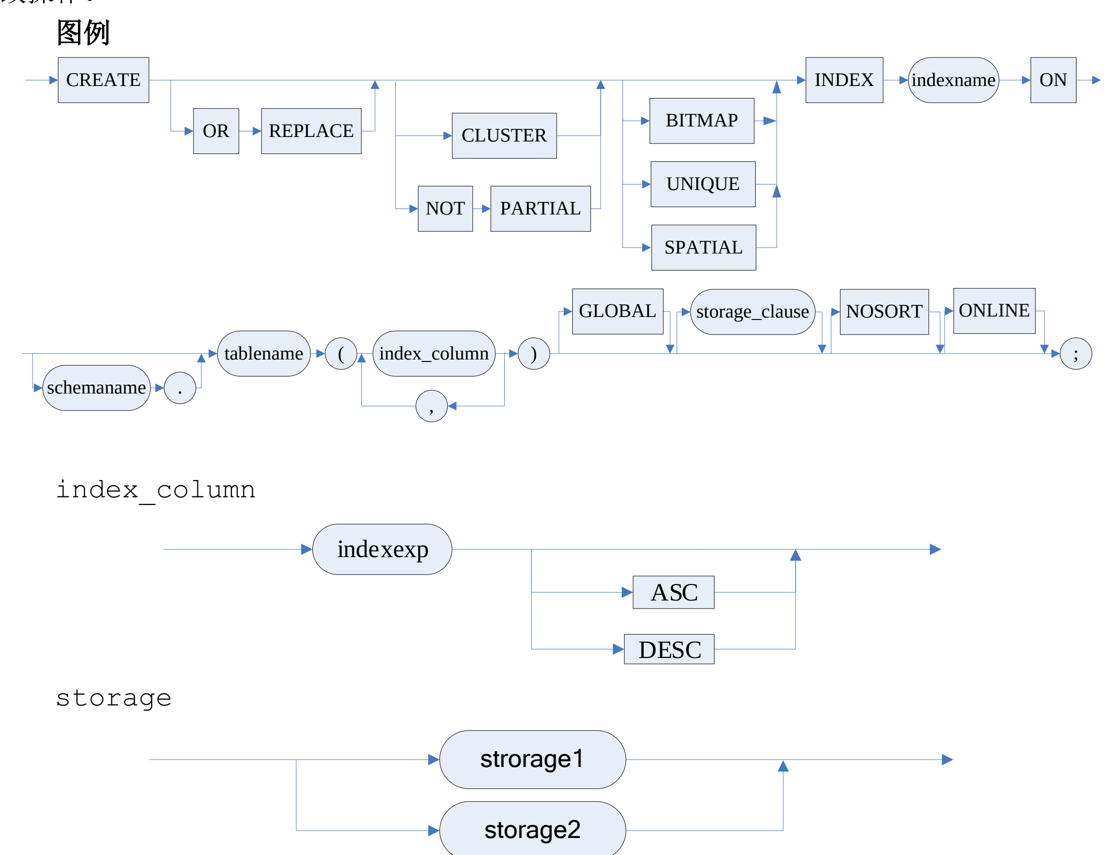
14.<STORAGE子句> 普通表的索引参考<STORAGE子句1>，HUGE表的索引参考<STORAGE子句2>；

15.<STORAGE子句1>中，BRANCH和NOBRANCH只能用以指定聚集索引；

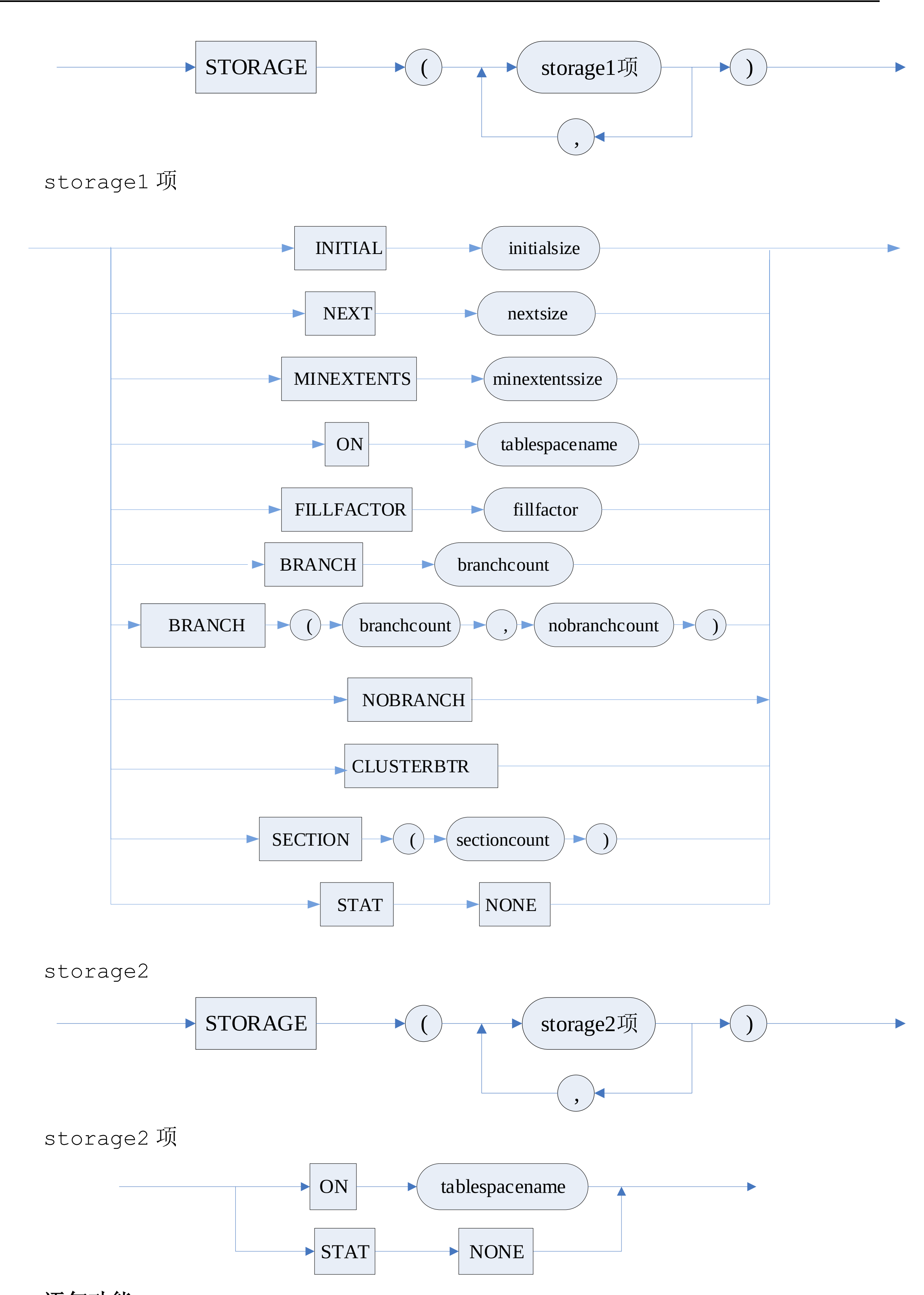
16.NOSORT 指明该索引相关的列已按照索引中指定的顺序有序，不需要在建索引时排序，提高建索引的效率。若数据非有序却指定了NOSORT，则在建索引时会报错；

17.ONLINE 表示支持异步索引，即创建索引过程中可以对索引依赖的表做增、删、改操作。

图例



storage1



语句功能

供 DBA 或该索引所属基表的拥有者且具有 CREATE INDEX 权限的用户定义索引。

使用说明

1. <索引名>不得与该模式中其它索引的名字相同；
2. 索引列不得重复出现且数据类型不得为多媒体数据类型、类类型和自定义类型；
3. 索引列最多不能超过63列；

4． 可以使用STORAGE子句指定索引的存储信息，它的参数说明参见CREATE TABLE语句；

5． 索引的默认表空间与其基表的表空间一致；

6． 索引的模式名与其基表的模式名一致；

7． 索引各字段值相加得到的记录内总数据值长度不得超过页大小的1/4，二级索引各字段值相加得到的记录内总数据值长度则不能超过min(页大小1/4 , 3000)；

8． 在下列情况下，DM利用索引可以提高性能：

1） 用指定的索引列值来搜索记录；

2） 用索引列的顺序来存取基表。

9． 每张表中只允许有一个聚集索引，如果之前已经指定过CLUSTER INDEX或者指定了CLUSTER PK，则用户新建立CLUSTER INDEX时系统会自动删除原先的聚集索引。但如果新建聚集索引时指定的创建方式（列，顺序）和之前的聚集索引一样，则会报错；

10．列存储表（HUGE表）和堆表不允许建立聚集索引；

11．指定CLUSTER INDEX操作需要重建表上的所有索引，包括PK索引；

12．删除聚集索引时，缺省以ROWID排序，自动重建所有索引；

13．本地索引：其分区方式与其所在基础表的分区方式一模一样的索引。本地索引的每个分区仅对应于其所在基础表的一个分区；

14．函数索引：创建方式与普通索引一样，并且支持UNIQUE和STORAGE设置项，对于以表达式为过滤的查询，创建合适的函数索引会提升查询效率；

函数索引具有以下约束：

1) 表达式可以由多列组成，不同的列不能超过 63 个；
2) 表达式里面不允许出现大字段和时间间隔类型列；
3) 不支持建立分区函数索引；
4) 函数索引表达式的长度理论值不能超过 816 个字符（包括生成后的指令和字符串）；
5) 函数索引不能为 CLUSTER 或 PRIMARY KEY 类型；
6) 表达式不支持集函数和不确定函数，不确定函数为每次执行得到的结果不确定，系统中不确定函数包括：RAND、SOUNDEX、CURDATE、CURTIME、CURRENT_DATE、CURRENT_TIME、CURRENT_TIMESTAMP、GETDATE、NOW、SYSDATE、CUR_DATABASE、DBID、EXTENT、PAGE、SESSID、UID、USER、VSIZE、SET_TABLE_OPTION、SET_INDEX_OPTION、UNLOCK_LOGIN、CHECK_LOGIN、GET_AUDIT、CFALGORITHMSENCRYPT、SF_MAC_LABEL_TO_CHAR、CFALGORITHMSDECRYPT、BFALGORITHMSENCRYPT、SF_MAC_LABEL_FROM_CHAR、BFALGORITHMSDECRYPT、SF_MAC_LABEL_CMP；
7) 快速装载不支持含有函数索引的表；
8) 当表中含有行前触发器并且该触发器会修改函数索引涉及列的值时，不能建立函数索引；
9) 若函数索引中要使用用户自定义的函数，则函数必须是指定了 DETERMINISTIC 属性的确定性函数；
10) 若函数索引中使用的确定性函数发生了变更或删除，用户需手动重建函数索引；
11) 若函数索引中使用的确定性函数内有不确定因素，会导致前后计算结果不同的情况。在查询使用函数索引时，使用数据插入函数索引时的计算结果为 KEY

值;修改时可能会导致在使用函数索引过程中出现根据聚集索引无法在函数索引中找到相应记录的情况，对此进行报错处理;

12) MPP 环境下不支持函数索引的创建。

15. 在水平分区表中建立唯一索引，分区键必须都包含在索引键中；不能对水平分区表建立唯一函数索引；不能对水平分区子表单独建立索引；

16. 非聚集索引和聚集索引不能使用OR REPLACE选项互相转换；

17. 位图索引：创建方式和普通索引一致，对低基数的列创建位图索引，能够有效提高基于该列的查询效率，位图索引具有以下约束：

1) 支持普通表、堆表和水平分区表创建位图索引；
2) 不支持对大字段创建位图索引；
3) 不支持对计算表达式列创建位图索引；
4) 不支持在 UNIQUE 列和 PRIMARY KEY 上创建位图索引；
5) 不支持对存在 CLUSTER KEY 的表创建位图索引；
6) 仅支持单列或者不超过 63 个组合列上创建位图索引；
7) MPP 环境下不支持位图索引的创建；
8) 不支持快速装载建有位图索引的表；
9) 不支持全局位图索引；
10) 包含位图索引的表不支持并发的插入、删除和更新操作。

18. NOSORT不支持与[OR REPLACE]、[CLUSTER]、[BITMAP]、[ONLINE]同用；

19. ONLINE选项具有以下约束：

1) 不支持与[OR REPLACE]、[CLUSTER]、[BITMAP]同用；
2) 不支持 MPP 环境；
3) 暂不支持列存储表创建索引时指定 ONLINE 选项；
4) 建立 PRIMARY KEY，UNIQUE 约束时，隐式创建的索引不支持 ONLINE 选项；
5) 函数索引支持 ONLINE 选项。

20. 空间索引：创建时需指定SPATIAL关键字，删除方式和普通索引一样。只能在DMGEO包内的空间类型的列上创建，如果使用DMGEO包内的空间位置进行查询时，使用空间索引能够提高查询效率。空间索引具有以下约束：

1) 只支持在空间类型列上创建；
2) 不支持使用 ONLINE 选项异步方式创建；
3) 不支持在水平分区表上创建空间索引；
4) 不支持在 MPP 环境和复制环境下创建空间索引；
5) 空间索引不支持组合索引，只能在一个列上创建；
6) 不支持在 4K 的页上建立空间索引。

举例说明

例 1 假设具有 DBA 权限的用户在 VENDOR 表中，以 VENDORID 为索引列建立索引 S1，以 ACCOUNTNO，NAME 为索引列建立唯一索引 S2。

```
CREATE  INDEX  S1  ON  PURCHASING.VENDOR (VENDORID);
CREATE UNIQUE INDEX S2 ON  PURCHASING.VENDOR (ACCOUNTNO, NAME);
```

例 2 假设具有 DBA 权限的用户在 SALESPERSON 表中，需要查询比去年销售额超过 20 万的销售人员信息，该过滤条件无法使用到单列上的索引，每次查询都需要进行全表扫描，效率较低。如果在 SALESTHISYEAR-SALESLASTYEAR 上创建一个函数索引，则可以较大程度提升查询效率。

```
CREATE INDEX INDEX_FBI ON SALES.SALESPERSON(SALESTHISYEAR-SALESLASTYEAR);
```

例 3 使用空间索引：

```
--创建含空间索引类型的表
DROP TABLE testgeo;
CREATE TABLE testgeo (id int, name varchar(20) , geo ST_polygon);
--创建空间索引
create spatial indeX spidx on testgeo (geo);
--删除空间索引 spidx
DROP INDEX spidx;
```

3.7.2 索引修改语句

为了满足用户在建立索引后还可随时修改索引的要求，DM 系统提供索引修改语句，包括修改索引名称、设置索引的查询计划可见性、改变索引有效性、重建索引和索引监控的功能。

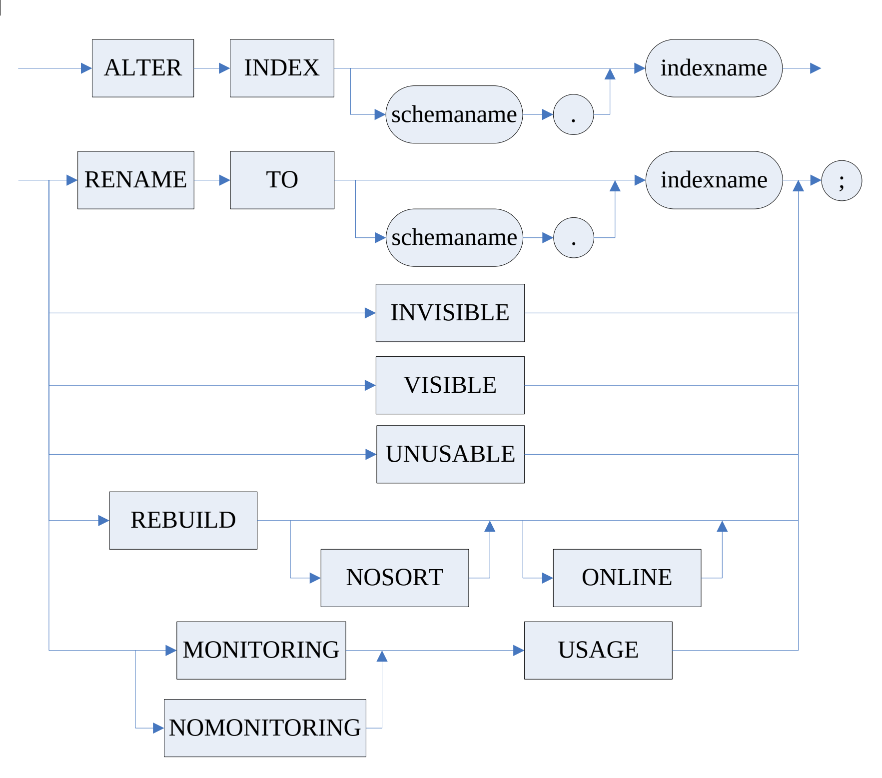
语法格式

```
ALTER  INDEX [<模式名>.]<索引名> <修改索引定义子句>
<修改索引定义子句> ::=
RENAME TO [<模式名>.]<索引名>|
<INVISIBLE | VISIBLE>
<UNUSABLE>|
<REBUILD>[NOSORT][ONLINE]|
<MONITORING | NOMONITORING> USAGE
```

参数

1. <模式名> 索引所属的模式，缺省为当前模式；
2. <索引名> 索引的名称。

图例



语句功能

1． 供具有 DBA 角色的用户或该索引所属基表的拥有者或具有 ALTER ANY INDEX 的用户修改索引；

2． 当索引修改成 INVISIBLE 时，查询语句的执行计划不会使用该索引，该索引相关的计划不会生成，用别的计划代替。修改成 VISIBLE，则会生成索引相关的计划。默认是 VISIBLE；

3． 当指定 UNUSABLE 时，索引将被置为无效状态，系统将不再维护此索引，若此索引被用于保证数据唯一性时，则此时不能够对表更新数据，可以在 SYSOBJECTS 的 VALID 字段查看该值。处于无效状态的索引，可以利用 REBUILD 来重建，或者先删除该索引再新建该索引。TRUNCATE 表会将该表所有失效索引置为生效；

4． 当指定 REBUILD 时，将置索引的状态为生效状态。NOSORT 指定重建时不需要排序，ONLINE 子句表示重建时使用异步创建逻辑，在过程中可以对索引依赖的表做增、删、改操作；

5． MONITORING USAGE 对指定索引进行监控；NOMONITORING USAGE 对指定索引取消监控。索引监控（MONITORING USAGE）仅支持对用户创建的二级索引进行监控，且不支持监控虚索引、系统索引、聚集索引、数组索引，相关监控信息可查看动态视图 V$OBJECT_USAGE。

使用说明

1． 使用者应拥有 DBA 权限或是该索引所属基表的拥有者；

2． <INVISIBLE | VISIBLE>仅支持表的二级索引修改，对聚集索引不起作用；

3． <UNUSABLE>仅支持对二级索引的修改，不支持位图索引、聚集索引、虚索引、水平分区子表和临时表上索引的修改；

4． <REBUILD>是 UNUSABLE 的逆向操作，支持和不支持修改的索引类型同 UNUSABLE；

5． 当 INI 参数 MONITOR_INDEX_FLAG 为 1 或 2 时，使用 ALTER INDEX 方式设置索引监控失效。

举例说明

例 1 具有 DBA 权限的用户需要重命名 S1 索引可用以下语句实现。

```
ALTER INDEX PURCHASING.S1 RENAME TO PURCHASING.S2;
```

例 2 当索引为 VISIBLE 时，查询语句执行计划如下：

```
CREATE TABLE TEST ( "C1" INT, "C2" INT);
CREATE  INDEX  INDEX_C1  ON TEST (C1);
explain  select c1 from test;
1   #NSET2: [0, 1, 12]
2     #PRJT2: [0, 1, 12]; exp_num(2), is_atom(FALSE)
3       #SSCN: [0, 1, 12]; INDEX_C1(TEST)
```

修改索引为 INVISIBLE 后，查询语句执行计划如下：

```
alter index  index_C1 INVISIBLE;
explain  select c1 from test;
1   #NSET2: [0, 1, 12]
2     #PRJT2: [0, 1, 12]; exp_num(2), is_atom(FALSE)
3       #CSCN2: [0, 1, 12]; INDEX33555442(TEST)
```

例 3 使用 UNUSABLE 将索引置为无效状态。

```
CREATE TABLE TEST ( "C1" INT, "C2" INT);
```

```
CREATE  INDEX  INDEX_C1  ON TEST (C1);
ALTER  INDEX  INDEX_C1  UNUSABLE;
```

此时系统将不维护 INDEX_C1，与此相关的计划均失效。注意，若索引是用于保证数据唯一性的，那么表仅能查询，不能更新。

```
CREATE  UNIQUE  INDEX  U_INDEX  ON  TEST(C2);
INSERT  INTO  TEST  VALUES(1,1);
```

将报“索引[U_INDEX]不可用”错误。

例 4 使用 REBUILD 将索引置为生效状态。

```
ALTER  INDEX  INDEX_C1  REBUILD;
```

3.7.3 索引删除语句

DM 系统允许用户在建立索引后还可随时删除索引。

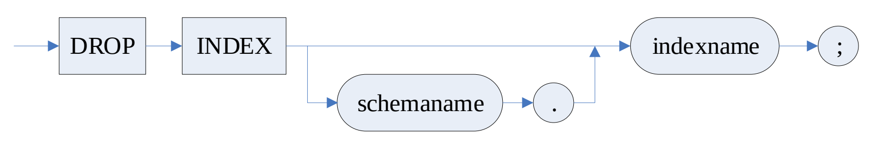
语法格式

```
DROP INDEX [<模式名>.]<索引名>;
```

参数

1. <模式名> 指明被删除索引所属的模式，缺省为当前模式；
2. <索引名> 指明被删除索引的名称。

图例



语句功能

供具有 DBA 角色的用户或该索引所属基表的拥有者删除索引。

使用说明

使用者应拥有 DBA 权限或是该索引所属基表的拥有者。

举例说明

例 具有 DBA 权限的用户需要删除 S2 索引可用以下语句实现。

```
DROP INDEX PURCHASING.S2;
```

3.8 管理位图连接索引

3.8.1 位图连接索引定义语句

位图连接索引是一种通过连接实现提高海量数据查询效率的有效方式，主要用于数据仓库环境中。它是针对两个或者多个表连接的位图索引，同时保存了连接的位图结果。对于列中的每一个值，该索引保存了索引表中对应行的 ROWID。

语法格式

```
CREATE [OR REPLACE] BITMAP INDEX <索引名>
ON bitmap_join_index_clause [<STORAGE 子句>];
bitmap_join_index_clause::=[<模式名>.]<表名>(<索引列定义>{,<索引列定义>})FROM [<模式名>.]<基表名>[别名]{,[<模式名>.]<基表名>[别名]} WHERE <条件表达式>;
```

```
<STORAGE 子句>请参考本章 3.6.1 节相关内容。
<索引列定义>::= [[<模式名>.]<表名|别名>]<索引列表达式>[ASC|DESC]
```

参数

1. OR REPLACE：只支持在第一次创建时使用 OR REPLACE，重建时不支持 OR REPLACE；

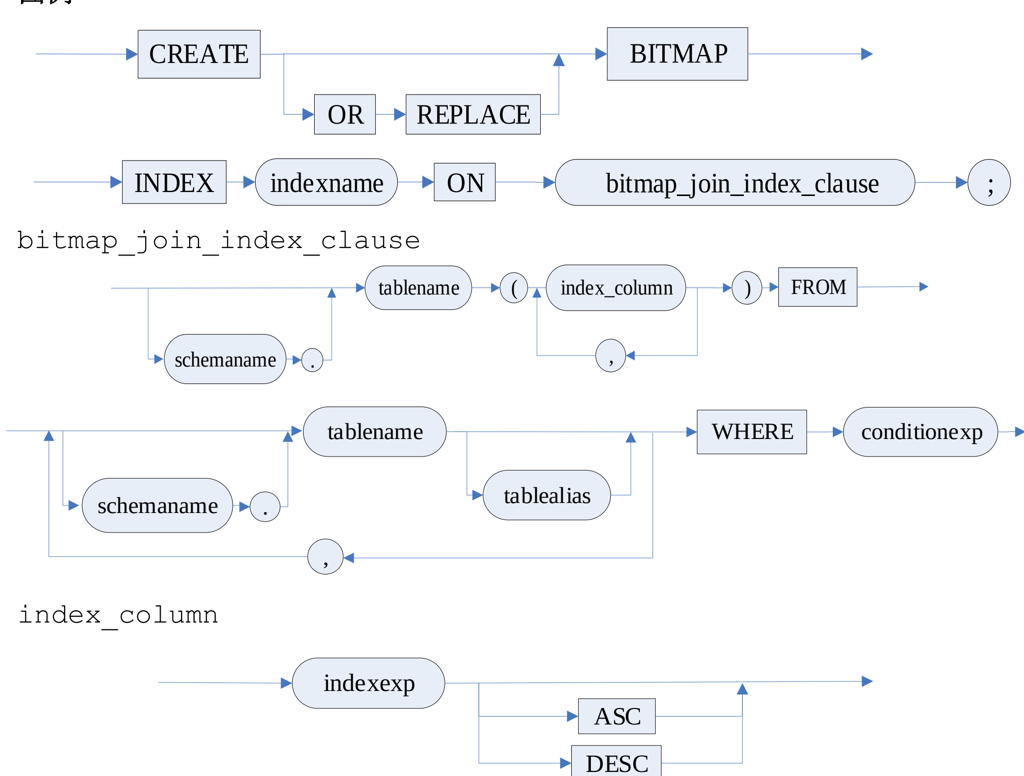
2. ON 子句：指定的表为事实表，括号内的列既可以是事实表的列也可以是维度表的列；

3. FROM 子句：指定参与连接的表；

4. WHERE 子句：指定连接条件；

5. 其他参数说明请参考本章 3.7 节。

图例



语句功能

供 DBA 或该索引所属基表的拥有者且具有 CREATE INDEX 权限的用户定义索引。

使用说明

1. 适用于常规索引的基本限制也适用于位图连接索引；

2. 用于连接的列必须是维度表中的主键或存在唯一约束；如果是复合主键，则必须使用复合主键中的所有列；

3. 当多个事务同时使用位图连接索引时，同一时间只允许更新一个表；

4. 连接索引创建时，基表只允许出现一次；

5. 不允许对存在 CLUSTER KEY 的表创建位图连接索引；

6. 位图连接索引表（内部辅助表，命名为 BMJ$_索引名）仅支持 SELECT 操作，其他操作都不支持：如 INSERT、DELETE、UPDATE、ALTER、DROP 和建索引等；

7. 不支持对位图连接索引所在事实表和维度表的备份还原，不支持位图连接索引表的表级备份还原；

8. 不支持位图连接索引表、位图连接索引以及虚索引的导出导入；

9. 位图连接索引及其相关表不支持快速装载；

10.位图连接索引名称的长度限制为：事实表名的长度+索引名称长度+6<128；

11.支持普通表、堆表和 HUGE 表；

12.WHERE 条件只能是列与列之间的等值连接，并且必须含有所有表；

13.事实表上聚集索引和位图连接索引不能同时存在；

14.不支持对含有位图连接索引的表中的数据执行 DML，如需要执行 DML，则先删除该索引；

15.含有位图连接索引的表不支持下列 DDL 操作：删除、修改表约束，删除、修改列，更改表名。另外，含位图连接索引的堆表不支持添加列操作；

16.不允许对含有位图连接索引的表并发操作；

17.创建位图连接索引时，在存储参数中可指定存储位图的字节数，有效值为：4~2048，服务器自动校正为 4 的倍数，默认值为 128。如 STORAGE(SECTION(4))，表示使用 4 个字节存储位图信息。

举例说明

创建位图连接索引：

```
create bitmap index SALES_CUSTOMER_NAME_IDX
on SALES.SALESORDER_HEADER(SALES.CUSTOMER.PERSONID)
from   SALES.CUSTOMER, SALES.SALESORDER_HEADER
where  SALES.CUSTOMER.CUSTOMERID = SALES.SALESORDER_HEADER.CUSTOMERID;
```

执行查询：

```
Select TOTAL
from SALES.CUSTOMER, SALES.SALESORDER_HEADER
where SALES.CUSTOMER.CUSTOMERID = SALES.SALESORDER_HEADER.CUSTOMERID
and SALES.CUSTOMER.PERSONID = '12';
```

3.8.2 位图连接索引删除语句

如果不需要位图连接索引可以使用删除语句删除。

删除（位图连接）索引语句格式：

```
DROP INDEX [<模式名>.]<索引名>;
```

参数

参数说明请参考本章 3.7 节。

举例说明

例如：

```
DROP INDEX sales.SALES_CUSTOMER_NAME_IDX;
```

3.9 管理全文索引

3.9.1 全文索引定义语句

用户可以在指定的表的文本列上建立全文索引。

语法格式

```
CREATE CONTEXT INDEX <索引名> ON [<模式名>.] <表名> (<索引列定义>) [<STORAGE 子句>] [LEXER <分词参数>] [<SYNC 子句>];
<索引列定义>, [<STORAGE 子句>] 请参考本章 3.7.1 节相关内容
<SYNC 子句> ::= SYNC [TRANSACTION]
```

参数

1． <索引名> 指明要创建的全文索引的名称，由于系统会为全文索引名加上前缀与后缀，因此用户指定的全文索引名长度不能超过122字节；

2． <模式名> 指明要创建全文索引的基表属于哪个模式，缺省为当前模式；

3． <表名> 指明要创建全文索引的基表的名称；

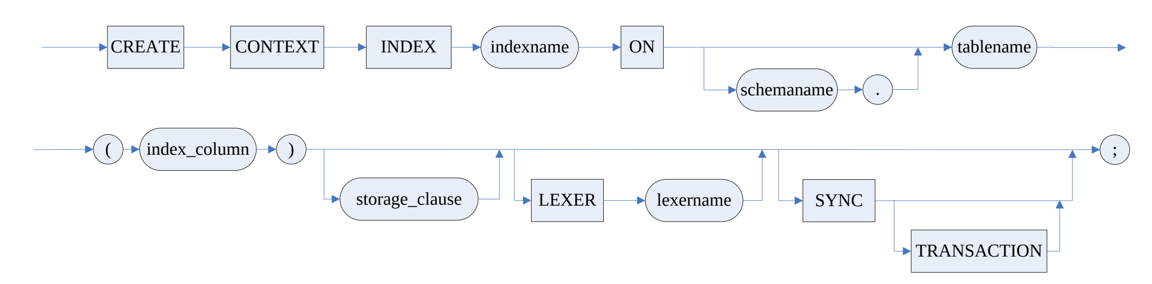
4． <列名> 指明基表中要创建全文索引的列的名称；

5． <分词参数> 指明全文索引分词器的分词参数；

6． <storage 子句> 只有指定表空间参数有效，其他参数无效（即 STORAGE ON xxx 或者 TABLESPACE xxx 有效，而诸如 INITIAL、NEXT 等无效）；

7． <SYNC 子句> 指明全文索引的同步类型。不指定<SYNC 子句>时创建全文索引后系统不进行全文索引填充；指定为 SYNC 时系统将在全文索引建立后对全文索引执行一次完全填充；指定为 SYNC TRANSACTION 时系统将在每次事务提交后，自动以增量更新方式填充全文索引，不需要用户手动填充。

图例



语句功能

供 DBA 或该全文索引基表的拥有者且具有 CREATE CONTEXT INDEX 权限的用户，在指定的表的文本列上建立全文索引。

使用说明

1． 全文索引必须在一般用户表上定义，而不能在系统表、视图、临时表、列存储表和外部表上定义；

2． 一个全文索引只作用于表的一个文本列，不允许为组合列和计算列；

3． 同一列只允许创建一个全文索引；

4． <列名>为文本列，类型可为CHAR、CHARACTER、VARCHAR、LONGVARCHAR、TEXT 或CLOB；

5． TEXT、CLOB类型的列可存储二进制字符流数据。如果用于存储DM全文检索模块能识别的格式简单的文本文件（如.txt，html等），则可为其建立全文索引；

6． 全文索引支持简体中文和英文；

7． 分词参数有5种：CHINESE_LEXER，中文最少分词；CHINESE_VGRAM_LEXER，机械双字分词，CHINESE_FP_LEXER，中文最多分词；ENGLISH_LEXER，英文分词；DEFAULT_LEXER，中英文最少分词，也是默认分词；

8． 创建全文索引时，若不指定<SYNC子句>，系统在创建全文索引后不进行填充，需要用户使用全文索引修改语句填充索引信息后才能进行全文检索；指定为SYNC时系统将在全文索引建立后对全文索引执行一次完全填充；指定为SYNC TRANSACTION时，每次事务

提交后，若基表数据发生变化，系统会自动以增量更新方式填充全文索引；

9. 不支持快速装载建有全文索引的表；

10. 不支持对建有全文索引的表进行表级备份恢复。

举例说明

例 用户 SYSDBA 需要在 PERSON 模式下的 ADDRESS 表的 ADDRES1 列上创建全文索引，可以用下面的语句：

```
CREATE CONTEXT INDEX INDEX0001 ON PERSON.ADDRESS(ADDRESS1) LEXER CHINESE_LEXER;
```

3.9.2 全文索引修改语句

全文索引需要根据基表进行索引数据填充，若基表数据变化而没有更新全文索引，可能引起全文检索结果不正确。

使用全文索引修改语句对全文索引进行完全填充和增量填充，使得全文索引的内容与表数据保持同步。

语法格式

```
ALTER CONTEXT INDEX <索引名> ON [<模式名>.] <表名> <REBUILD | INCREMENT | OPTIMIZE>[ONLINE] [LEXER <分词参数>];
```

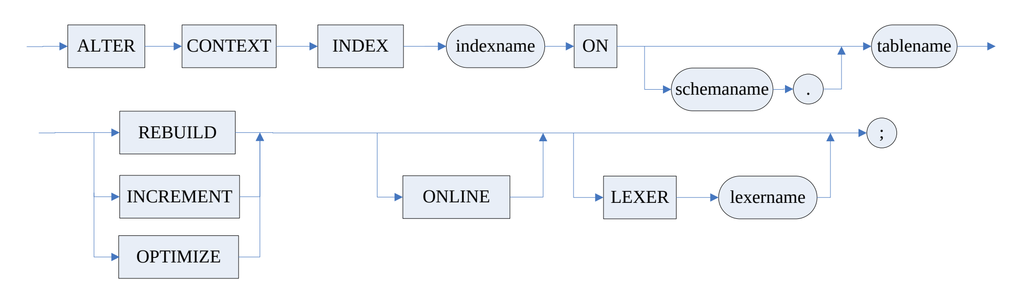
参数

1. <索引名> 指明被操作的全文索引的名称；

2. <模式名> 指明被操作的全文索引属于哪个模式，缺省为当前模式；

3. <表名> 指明被操作的基表的名称。

图例



语句功能

REBUILD 为完全填充，此方式首先会将全文索引辅助清空，再将基表中所有记录逐个取出，根据分词算法获得分词结果，即字/词所在记录的 ROWID 和出现次数，保存在词表中。INCREMENT 为增量填充，此方式只是将基表中发生数据变化的记录执行分词并保存分词结果。OPTIMIZE 操作仅仅对全文索引辅助表进行优化，去除冗余信息，不影响查询结果。

使用说明

1. 若未使用 SYNC 选项创建全文索引，则需要以 REBUILD 方式调用此语句，然后才能进行全文检索。这时该修改语句起到填充全文索引信息的作用；

2. 当该列数据更新后，为了对更新后的数据进行检索，需要再次填充全文索引信息，以确保查询的正确性；

3. DM 服务器启动时，不会自动加载词库，而是在第一次执行全文索引填充时加载，之后直到服务器停止才释放；

4. 在完全更新全文索引后，如果表数据发生少量更新，利用 INCREMENT 增量填充方式更新全文索引可以提高效率；

5． 语句中指定 ONLINE 选项时，指明对全文索引进行异步填充，允许同时对全文索引所在的表进行增删改操作；

6． LEXER 子句只能与 REBUILD 方式一起使用。

举例说明

例 用户 SYSDBA 需要在 PERSON 模式下的 ADDRESS 表的 ADDRES1 列上完全填充全文索引，可以用下面的语句：

```
ALTER CONTEXT INDEX INDEX0001 ON PERSON.ADDRESS REBUILD;
```

3.9.3 全文索引删除语句

删除全文索引。

语法格式

```
DROP CONTEXT INDEX <索引名> ON [<模式名>.] <表名>;
```

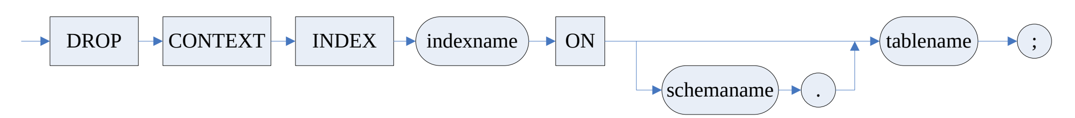
参数

1． <索引名> 指明被操作的全文索引的名称；

2． <模式名> 指明被操作的全文索引属于哪个模式，缺省为当前模式；

3． <表名> 指明被操作的基表的名称。

图例



语句功能

供具有全文索引删除权限的用户或该全文索引所属基表的拥有者删除全文索引，包括删除数据字典中的相应信息和全文索引内容。

使用说明

1． 除了该语句可删除全文索引外，当数据库模式发生如下改变时，系统将自动调用全文索引删除模块：

  1） 删除表时，删除表上的全文索引；

  2） 删除建立了全文索引的列时，删除列上的全文索引；

2． 不允许修改建有全文索引的列。

举例说明

例 用户 SYSDBA 需要删除在 PERSON 模式下 ADDRESS 表的全文索引，可以用下面的语句：

```
DROP CONTEXT INDEX INDEX0001 ON PERSON.ADDRESS;
```

3.10 管理空间索引

空间索引创建与删除请参考 3.7 节。

空间数据并不能直接进行比较，根据空间数据查询大多是根据空间函数进行查询。若要使用空间索引，需要满足如下条件：

1） 使用 DMGEO 包内的空间函数作为查询条件，当前能够使用到空间索引的函数有：DMGEO.ST_WITHIN 、 DMGEO.ST_DISTANCE 、 DMGEO.ST_DISJOINT 、

DMGEO.ST_EQUALS 、 DMGEO.ST_TOUCHES 、 DMGEO.ST_OVERLAPS 、 DMGEO.ST_CROSSES 、 DMGEO.ST_INTERSECTS 、 DMGEO.ST_CONTAINS 、 DMGEO.ST_RELATE 等，具体可参考《DM8 系统包使用手册》中 DMGEO 包的介绍；

2） 空间函数的第一个参数必须是空间索引的列；

3） 空间函数中与之比较的空间数据必须是常量或固定的值；

4） 对于 ST_DISTANCE，仅支持<和<=某个常量值的条件；

5） 对于其他的返回 1 和 0 表示 TRUE 和 FALSE 的函数，只支持缺省比较条件或=1 的比较条件。

举例说明

```
--查询表中被指定空间对象包含的数据
select * from testgeo where dmgeo.ST_WITHIN(geo, dmgeo.ST_GEOMFromText ( 'polygon ((10 10, 10 20, 20 20, 20 15, 10 10))' , 4269 ) ) = 1;
或
select * from testgeo where dmgeo.ST_WITHIN(geo, dmgeo.ST_GEOMFromText ( 'polygon ((10 10, 10 20, 20 20, 20 15, 10 10))' , 4269 ) ) ;
--查询表中据制定空间对象距离小于 10 的数据
select * from testgeo where dmgeo.ST_DISTANCE(geo, dmgeo.ST_GEOMFromText ( 'polygon ((10 10, 10 20, 20 20, 20 15, 10 10))' , 4269 ) ) < 10;
```

3.11 管理数组索引

数组索引指在一个只包含单个数组成员的对象列上创建的索引。

3.11.1 数组索引定义语句

语法格式

```
CREATE ARRAY INDEX <索引名> ON [<模式名>.] <表名> (<索引列定义>)
```

使用说明

1． 暂不支持在水平分区表上创建数组索引；

2． 暂时不支持在有数组索引表上进行批量装载(数组索引失效的例外)；

3． 支持创建数组索引的对象只能包含数组一个成员。数组可以是 DM 静态数组、动态数组或者 ORACLE 兼容的嵌套表或 VARRAY；

4． 数组项类型只能是可比较的标量类型，不支持复合类型、对象类型或大字段类型；

5． 临时表不支持；

6． 数组索引不支持改名；

7． 数组索引列不支持改名；

8． 数组索引只能是单索引，不能为组合索引；

9． 不支持空值的检索

10．MPP 环境不支持数组索引。

3.11.2 数组索引修改语句

数组索引修改语句与普通索引用法相同，请参考 3.7 节。与普通索引不同的是，数组索引不支持 NOSORT 和 ONLINE 用法。

3.11.3 数组索引使用

使用数组索引进行查询，必须使用谓词 CONTAINS。

语法格式

```
CONTAINS(<索引列名>,<val{,val}>)
或者
CONTAINS(<索引列名>,arr_var_exp)
```

参数

1. val：必须为与对象列数组项相同或可转换的标量类型表达式。

2. arr_var_exp：必须为数组类型（DM 静态数组、动态数组或者 ORACLE 兼容的嵌套表或 VARRAY），其数组项类型必须与对象列数组项类型相同或可转换。

举例说明

```
CREATE TYPE ARR_NUM1 IS VARRAY(1024) OF NUMBER;  --VARRAY 数组
/
CREATE TYPE ARR_NUM2 IS TABLE OF NUMBER;  --嵌套表
/
CREATE TYPE ARR_NUM3 IS ARRAY NUMBER[];  --动态
/
CREATE TYPE ARR_NUM4 IS ARRAY NUMBER[3]; --静态
/
CREATE CLASS CLS1 AS V ARR_NUM1;END;
/

CREATE TABLE TEST (C1 CLS1);
INSERT INTO TEST VALUES(CLS1(ARR_NUM1(1,2,3)));
INSERT INTO TEST VALUES(CLS1(ARR_NUM1(1,2)));
INSERT INTO TEST VALUES(CLS1(ARR_NUM1(2,1)));
INSERT INTO TEST VALUES(CLS1(ARR_NUM1(1,5)));
INSERT INTO TEST VALUES(CLS1(ARR_NUM1(2,4)));
INSERT INTO TEST VALUES(CLS1(ARR_NUM1(4,5,6)));

CREATE ARRAY INDEX IDX ON TEST(C1);   --创建数组索引
SELECT * FROM TEST WHERE CONTAINS(C1,1,2,3);  --使用数组索引查询

--嵌套表
DECLARE
```

```
X    ARR_NUM2;
BEGIN
X := ARR_NUM2();
X.EXTEND(3);
X(1) := 1;
X(2) := 2;
X(3) := 3;
SELECT * FROM TEST WHERE CONTAINS(C1,X);
END;
/

--动态数组
DECLARE
X    ARR_NUM3;
BEGIN
X := NEW NUMBER [3];
X[1]:= 1;
X[2]:= 2;
X[3]:= 3;
SELECT * FROM TEST WHERE CONTAINS(C1,X);
END;
/

--静态数组
DECLARE
X    ARR_NUM4;
BEGIN
X[1]:= 1;
X[2]:= 2;
X[3]:= 3;
SELECT * FROM TEST WHERE CONTAINS(C1,X);
END;
/
```

3.11.4 数组索引删除语句

数组索引删除语句与普通索引用法相同，请参考 3.7 节。

3.12 管理序列

3.12.1 序列定义语句

序列是一个数据库实体，通过它多个用户可以产生唯一整数值，可以用序列来自动地生成主关键字值。

语法格式

```
CREATE SEQUENCE [ <模式名>.] <序列名>  [ <序列选项列表>];
<序列选项列表> ::=  <序列选项>{<序列选项>}
<序列选项> ::=
    INCREMENT BY <增量值>|
    START WITH <初值>|
    MAXVALUE <最大值>|
    NOMAXVALUE|
    MINVALUE <最小值>|
    NOMINVALUE|
    CYCLE|
    NOCYCLE|
    CACHE <缓存值>|
    NOCACHE|
    ORDER |
    NOORDER |
    GLOBAL |
    LOCAL
```

参数

1． <模式名> 指明被创建的序列属于哪个模式，缺省为当前模式；

2． <序列名> 指明被创建的序列的名称，序列名称最大长度128字节；

3． <增量值> 指定序列数之间的间隔，这个值可以是[-1000000, 1000000]之间任意的DM正整数或负整数，但不能为0。如果此值为负，序列是下降的，如果此值为正，序列是上升的。如果忽略INCREMENT BY子句，则间隔缺省为1。增量值的绝对值必须小于(<最大值> - <最小值>)；

4． <初值> 指定被生成的第一个序列数，可以用这个选项来从比最小值大的一个值开始升序序列或比最大值小的一个值开始降序序列。对于升序序列，缺省值为序列的最小值，对于降序序列，缺省值为序列的最大值；

5． <最大值> 指定序列能生成的最大值，如果忽略MAXVALUE子句，则降序序列的最大值缺省为-1，升序序列的最大值缺省为9223372036854775807(0x7FFFFFFFFFFFFFFF)，若指定的最大值超出缺省最大值，则DM自动将最大值置为缺省最大值。非循环序列在到达最大值之后，将不能继续生成序列数；

6． <最小值> 指定序列能生成的最小值，如果忽略MINVALUE子句，则升序序列的最小值缺省为1，降序序列的最小值缺省为-9223372036854775808(0x8000000000000000)，若指定的最小值超出缺省最小值，则DM自动将最小值置为缺省最小值。循环序列在到达最小值之后，将不能继续生成序列数。

最小值必须小于最大值；

7. CYCLE 该关键字指定序列为循环序列：当序列的值达到最大值/最小值时，序列将从最小值/最大值计数；

8. NOCYCLE 该关键字指定序列为非循环序列：当序列的值达到最大值/最小值时，序列将不再产生新值；

9. CACHE 该关键字表示序列的值是预先分配，并保持在内存中，以便更快地访问；<缓存值>指定预先分配的值的个数，最小值为2；最大值为50000；且缓存值不能大于(<最大值> - <最小值>)/<增量值>；

10. NOCACHE 该关键字表示序列的值是不预先分配；

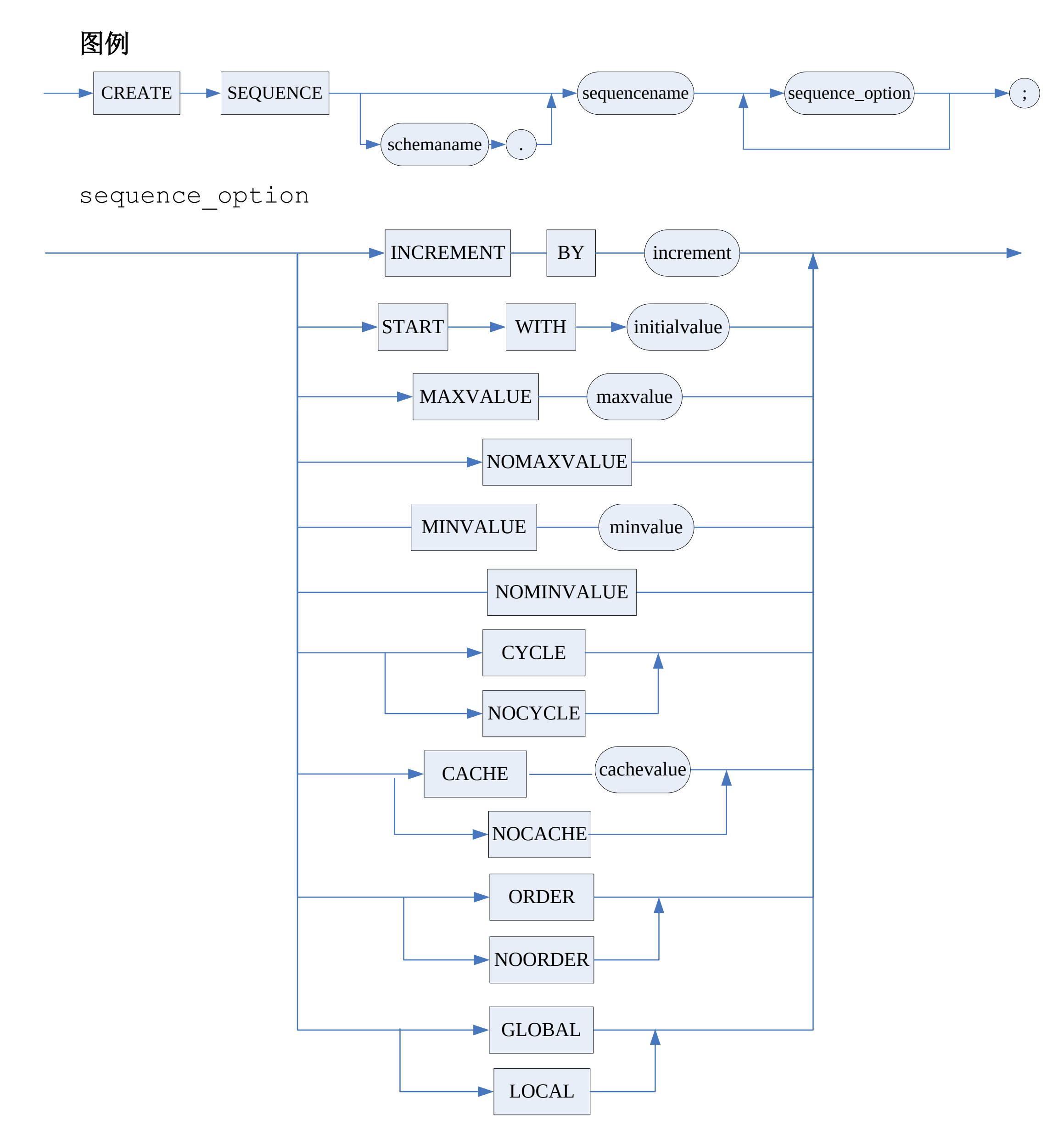
11. ORDER 该关键字表示以保证请求顺序生成序列号；

12. NOORDER 该关键字表示不保证请求顺序生成序列号；

13. GLOBAL 该关键字表示MPP环境下序列为全局序列，缺省为GLOBAL；

14. LOCAL 该关键字表示MPP环境下序列为本地序列。

图例



语句功能

创建一个序列。只有 DBA 或该序列的拥有者且具有 CREATE SEQUENCE 权限的用户才能创建序列。

使用说明

1． 一旦序列生成，就可以在 SQL 语句中用以下伪列来存取序列的值；

  1） CURRVAL 返回当前的序列值；

  2） NEXTVAL 如果为升序序列，序列值增加并返回增加后的值；如果为降序序列，序列值减少并返回减少后的值。如果第一次对序列使用该函数，则返回序列当前值；

  3） 用户会话在第一次使用 CURRVAL 之前应先使用 NEXTVAL 获取序列当前值；之后除非会话使用 NEXTVAL 获取序列当前值，否则每次使用 CURRVAL 返回的值不变。

2． 缺省序列：如果在序列中什么也没有指出则缺省生成序列，一个从 1 开始增量为 1 且无限上升(最大值为 9223372036854775807)的升序序列；仅指出 INCREMENT BY -1，将创建一个从-1 开始且无限下降(最小值为-9223372036854775808)的降序序列；

3． LOCAL 类型序列的最高 10 位用来记录标识 MPP 节点号，因此 LOCAL 类型的序列值和 GLOBAL 类型序列在范围、最大值、最小值上都有所差别。LOCAL 类型序列创建时可设置的最大值、最小值分别为 9007199254740991、-9007199254740992。需要注意的是，最高 10 位设置了 MPP 站点号以后，序列的真实值实际上可能不会在序列定义的最大最小值范围内。

举例说明

例 创建序列 SEQ_QUANTITY，将序列的前两个值插入表 PRODUCTION.PRODUCT_INVENTORY 中。

```
(1)创建序列 SEQ_QUANTITY
CREATE  SEQUENCE  SEQ_QUANTITY  INCREMENT  BY 10;
(2)将序列的第一个值插入表 PRODUCT_INVENTORY 中
INSERT   INTO   PRODUCTION.PRODUCT_INVENTORY   VALUES(1,1, SEQ_QUANTITY.NEXTVAL);
SELECT  *  FROM  PRODUCTION.PRODUCT_INVENTORY;
查询结果为：表 PRODUCT_INVENTORY 增加一行，列 QUANTITY 的值为 1
(3)将序列的第二个值插入表 PRODUCT_INVENTORY 中
INSERT   INTO   PRODUCTION.PRODUCT_INVENTORY   VALUES(1,1, SEQ_QUANTITY.NEXTVAL);
SELECT  *  FROM  PRODUCTION.PRODUCT_INVENTORY;
查询结果为：表 PRODUCT_INVENTORY 增加两行，列 QUANTITY 的值分别为 1，11
```

3.12.2 序列修改语句

DM 系统提供序列修改语句，包括修改序列步长值、设置序列最大值和最小值、改变序列的缓存值、循环属性、ORDER 属性。

语法格式

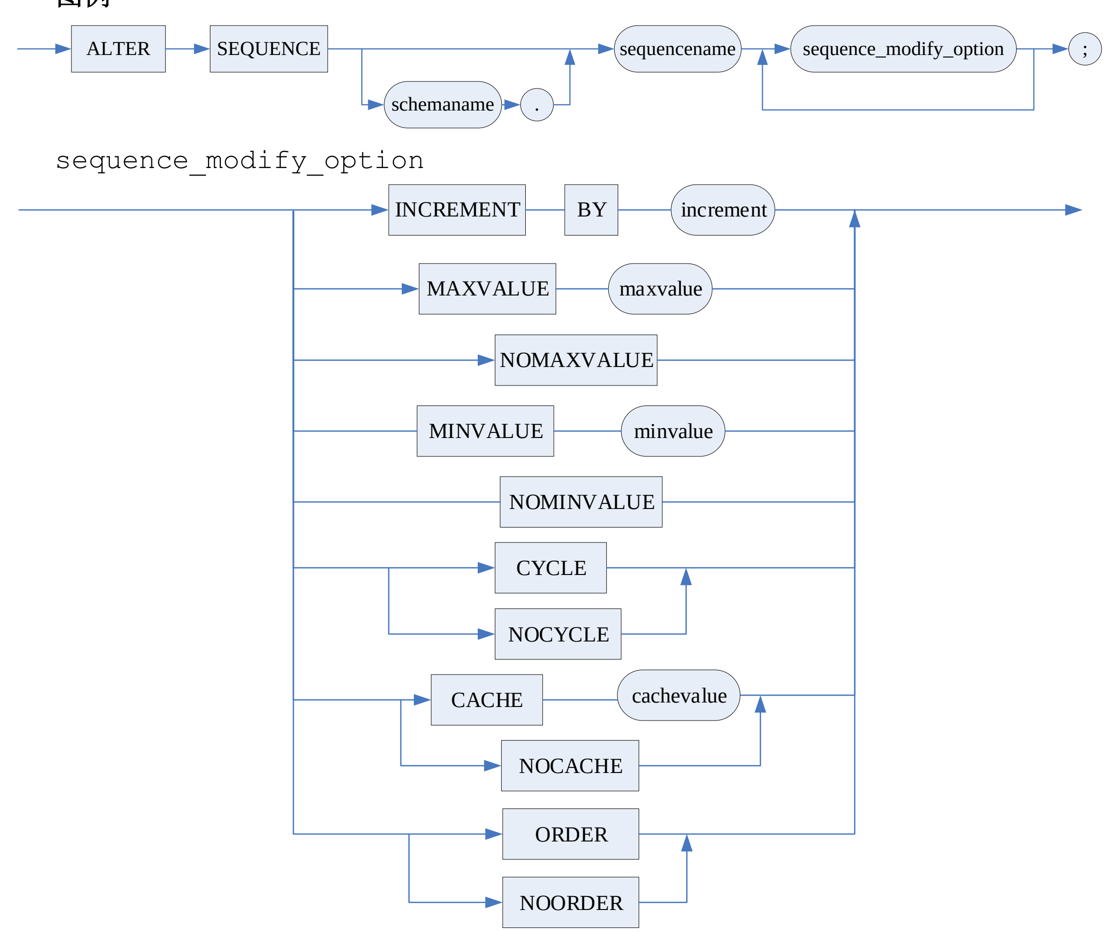
```
ALTER  SEQUENCE [ <模式名>.] <序列名>  [ <序列修改选项列表>];
<序列选项列表> ::=  <序列修改选项>{<序列修改选项>}
<序列修改选项> ::=
    INCREMENT BY <增量值>|
```

```
MAXVALUE <最大值>|
NOMAXVALUE|
MINVALUE <最小值>|
NOMINVALUE|
CYCLE|
NOCYCLE|
CACHE <缓存值>|
NOCACHE|
ORDER|
NOORDER
```

参数

与序列定义语句的参数相同。

图例



语句功能

修改一个序列，只有 DBA 或该序列的拥有者或具有 ALTER ANY SEQUENCE 权限的用户才修改序列。

使用说明

1. 关于步长的修改，分两种情况：
   a) 如果在修改前没有用 NEXTVAL 访问序列，创建完序列后直接修改序列步长值，则序列的当前值为起始值加上新步长值与旧步长值的差；
   b) 如果在修改前用 NEXTVAL 访问了序列，然后修改序列步长值，则再次访问序列的当前值为序列的上一次的值加上新步长值。

2． 缺省序列选项：如果在修改序列语句中没有指出某选项则缺省是修改前的选项值。不允许未指定任何选项、禁止重复或冲突的选项说明；

3． 序列的起始值不能修改；

4． 修改序列的最小值不能大于起始值、最大值不能小于起始值；

5． 修改序列的步长的绝对值必须小于MAXVALUE与MINVALUE的差；

6． 序列的当前值不能大于最大值，不能小于最小值。

举例说明

例1 创建完序列后直接修改序列的步长。

```
    CREATE SEQUENCE SEQ1  INCREMENT BY 1000  START WITH 5 NOMAXVALUE NOMINVALUE
CACHE 10;
    ALTER SEQUENCE SEQ1 INCREMENT BY 1 ;
    SELECT SEQ1.NEXTVAL FROM DUAL;
```

查询结果为：-994

例2 创建序列后使用NEXTVAL访问了序列，然后修改步长。

```
    CREATE SEQUENCE SEQ2   INCREMENT BY 1000  START WITH 5 NOMAXVALUE NOMINVALUE
NOCACHE ;
    SELECT SEQ2.NEXTVAL FROM DUAL;
    ALTER SEQUENCE SEQ2 INCREMENT BY 1 ;
    SELECT SEQ2.NEXTVAL FROM DUAL;
```

查询结果为：6

例3 修改序列的最小值。

```
    CREATE SEQUENCE SEQ3   INCREMENT BY 1  START WITH 100   MINVALUE  3 ;
    ALTER SEQUENCE SEQ3   MINVALUE  2;
```

3.12.3 序列删除语句

DM 系统允许用户在建立序列后还可随时删除序列。

语法格式

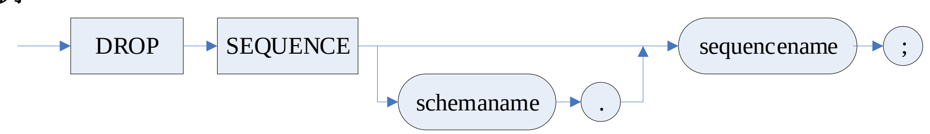
```
    DROP SEQUENCE [ <模式名>.]<序列名>;
```

参数

1． <模式名> 指明被删除序列所属的模式，缺省为当前模式；

2． <序列名> 指明被删除序列的名称。

图例



语句功能

供具有 DBA 角色的用户或该序列的拥有者从数据库中删除序列生成器。

使用说明

一种重新启动序列生成器的方法就是删除它然后再重新创建，例如有一序列生成器当前值为 150，而且用户想要从值 27 开始重新启动此序列生成器，他可以：

1） 删除此序列生成器；

2） 重新以相同的名字创建序列生成器，START WITH 选项值为 27。

举例说明

例 用户 SYSDBA 需要删除序列 SEQ_QUANTITY，可以用下面的语句：

```
DROP SEQUENCE SEQ_QUANTITY;
```

3.13 管理 SQL 域

为了支持 SQL 标准中的域对象定义与使用，DM 支持 DOMAIN 的创建、删除以及授权 DDL 语句，并支持在表定义中使用 DOMAIN。

域（DOMAIN）是一个可允许值的集合。域在模式中定义，并由<域名>标识。域是用来约束由各种操作存储于基表中某列的有效值集。域定义说明一种数据类型，它也能进一步说明约束域的有效值的<域约束>，还可说明一个<缺省子句>，该子句规定没有显式指定值时所要用的值或列的缺省值。

3.13.1 创建 DOMAIN

CREATE DOMAIN 创建一个新的数据域。 定义域的用户成为其所有者。DOMAIN 为模式类型对象，其名称在模式内唯一。

语法格式

```
CREATE DOMAIN <domain name> [ AS ] <数据类型> [ <default clause> | <domain constraint>] ;
<domain constraint>::=[<constraint name definition>]<check constraint definition>
<constraint name definition>::=CONSTRAINT <约束名>
<check constraint definition>::= CHECK (expression)
```

参数

1. <domain name> 要创建的域名字（可以有模式前缀）。如果在 CREATE SCHEMA 语句中定义 DOMAIN，则<domain name>中的模式前缀（如果有）必须与创建的模式名一致。

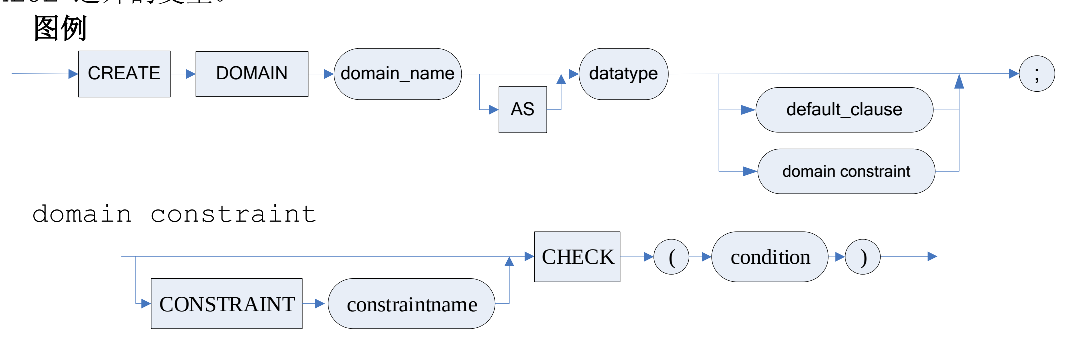
2. <data type> 域的数据类型。仅支持定义标准的 SQL 数据类型。

3. <default clause> DEFAULT 子句为域数据类型的字段声明一个缺省值。 该值是任何不含变量的表达式（但不允许子查询）。 缺省表达式的数据类型必需匹配域的数据类型。如果没有声明缺省值， 那么缺省值就是空值。缺省表达式将用在任何为该字段声明数值的插入操作。 如果为特定的字段声明了缺省值，那么它覆盖任何和该域相关联的缺省值。

4. <constraint name definition> 一个约束的可选名称。如果没有名称，系统生成一个名字。

5. <check constraint definition> CHECK 子句声明完整性约束或者是测试，域的数值必须满足这些要求。 每个约束必须是一个生成一个布尔结果的表达式。它应该使用名字 VALUE 来引用被测试的数值。CHECK 表达式不能包含子查询，也不能引用除 VALUE 之外的变量。

图例



语句功能

供具有 CREATE DOMAIN 和 CREATE ANY DOMAIN 系统权限的用户创建 DOMAIN。

举例说明

```
CREATE DOMAIN DA INT CHECK (VALUE < 100);
```

3.13.2 使用 DOMAIN

在表定义语句中，支持为表列声明使用域。如果列声明的类型定义使用域引用，则此列定义直接继承域中的数据类型、缺省值以及 CHECK 约束。如果列定义使用域，然后又自己定义了缺省值，则最终使用自己定义的缺省值。

用户可以使用自己的域。如果要使用其它用户的域，则必须被授予了该域的 USAGE 权限。DBA 角色默认拥有此权限。

例 在 T 表中使用第 1 节中创建的域 DA。

```
CREATE TABLE T(ID DA);
```

列定义虽然使用了域，但其 SYSCOLUMNS 系统表中类型相关字段记录域定义的数据类型，也就是说，从 SYSCOLUMNS 系统表中不会表现出对域的引用。

使用说明

使用某个域的用户必须具有该域的 USAGE DOMAIN 或 USAGE ANY DOMAIN 权限。

3.13.3 删除 DOMAIN

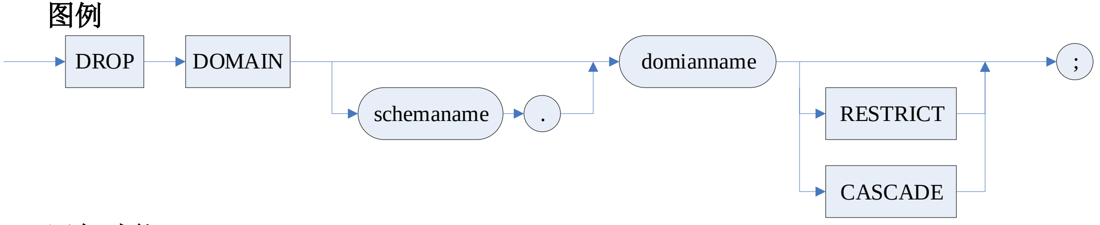
语法格式

```
DROP DOMAIN <domain name> <drop behavior>;
<drop behavior> ::=
    RESTRICT|
    CASCADE
```

参数

RESTRICT 表示仅当 DOMAIN 未被表列使用时才可以被删除； CASCADE 表示级联删除。

图例



语句功能

删除一个用户定义的域。用户可以删除自己拥有的域，具有 DROP ANY TABLE 系统权限的用户则可以删除任意模式下的域。

举例说明

```
DROP DOMAIN DA CASCADE;
```

3.14 管理上下文

CONTEXT 上下文提供了一组设置以及访问服务器运行时应用数据的接口，通过这些接口可以有效控制应用数据的修改和访问。

CONTEXT 上下文有一个库级唯一的标识：名字空间（NAMESPACE），该名字空间保存了（NAME-VALUE）格式的数据，该数据通过 DBMS_SESSION 包中的过程设置其值，并通

过 SYS_CONTEXT 系统函数访问其值。

应用上下文可以访问当前会话的一些属性信息，例如当前会话的 ID、模式名、用户名等。在实际应用过程中，将这些信息添加到查询的过滤条件中，起到允许或者禁止某些用户访问这些应用数据的目的。

3.14.1 创建上下文

创建上下文。

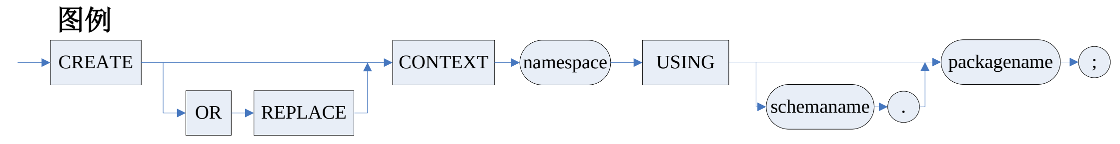
语法格式

```
CREATE [OR REPLACE] CONTEXT <namespace> USING [<模式名>.] <packagename>;
```

参数

1. OR REPLACE 使用一个不同的 package 重新定义 namespace；
2. <namespace> 上下文的名字空间，保存到系统表 sysobjects 中，其模式为 SYS；
3. <模式名> 上下文关联的 package 的模式，默认为当前模式；
4. <packagename> 上下文关联的包，创建上下文不会影响已有的 package；

图例



语句功能

创建一个上下文 namespace，namespace 通过关联的 DMSQL 程序包来管理上下文的属性和值。属性和值就是用户的会话信息。

使用说明

1. 具有 CREATE ANY CONTEXT 权限的用户，才可以创建上下文；
2. 通过 DBMS_SESSION 包的 SET_CONTEXT 过程设置上下文的属性和值，具体可参考《DM8 系统包使用手册》第 17 章；
3. 通过系统函数 SYS_CONTEXT 访问 CONTEXT namespace 的属性值。其语法如下：

```
SYS_CONTEXT('namespace', 'parameter' [, length ])
```

使用说明如下：

1) namespace：已创建的 namespace，不区分大小写，如果不存在，则报错；
2) parameter：namespace 中的属性，不区分大小写，如果不存在，则返回 NULL；最大长度 30bytes；
3) 函数返回值类型为 varchar(256)，如果返回值超过 256 字节，则需要设置 length。length 为 INT 类型，最大值为 4000，即返回字符串最大长度为 4000 字节；如果指定的 length 值非法，则忽略该值，而使用默认值 256。

4. 会话建立之后，只能通过关联的 package 来修改 namespace 内的属性值。namespace 中的属性值只允许会话级访问，即只有设置其属性值的会话才可以访问该值，其他会话访问该属性值都为空。namespace 保存的属性值为系统运行时的动态数据，服务器重启后该数据会丢失；
5. 成功创建上下文后，其信息保存在系统表 SYSOBJECTS 中，详见下表：

表 3.14.1 CONTEXT 与系统表 SYSOBJECTS 字段映射关系

字段	名称	类型	说明
name	NAMESPACE	VARCHAR2(30)	CONTEXT 名字空间
SCHID	SCHID	INT	CONTEXT 所在模式 ID，即 SYS 模式 ID
info5	SCHEMA	VARCHAR2(30)	关联包所在模式名

info6	PACKAGE	VARCHAR2(30)	关联包名
TYPE$	TYPE	VARCHAR2(10)	SCHOBJ
SUBTYPE$	SUBTYPE$	VARCHAR2(10)	CONTEXT

6. USERENV 为系统默认的上下文名字空间，保存了用户的上下文信息，属性如下：

表 3.14.2 USERENV 属性

属性	说明
CURRENT_SCHEMA	返回当前模式名
CURRENT_SCHEMAID	返回当前模式 ID
CURRENT_USER	返回当前的用户名
CURRENT_USERID	返回当前的用户 ID
DB_NAME	返回数据名
HOST	返回客户端的主库名
INSTANCE_NAME	返回实例名
IP_ADDRESS	返回客户端的 IP 地址
ISDBA	如果当前会话用户拥有 DBA 权限，则返回 TRUE，否则，返回 FALSE
LANG	语言包简写，中文返回"CN"，英文返回"EN"
LANGUAGE	语言包，返回库的编码方式
NETWORK_PROTOCOL	通信协议
SERVER_HOST	实例运行的主机名
SESSION_USER	会话的用户名
SESSION_USERID	会话的用户 ID
SID	当前会话的 ID

7. 动态视图 V$CONTEXT 显示当前会话所有上下文的名字空间、属性和值。

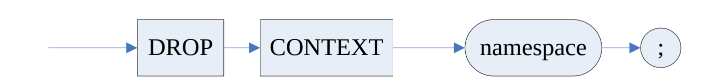
3.14.2 删除上下文

删除上下文。

语法格式

```
DROP CONTEXT <namespace> ;
```

图例



语句功能

从数据库中删除上下文的 namespace，即从系统表 sysobjects 中删除。如果 namespace 不存在，则报错。如果该 namespace 成功删除，但之前已添加了属性和值，那么会话仍可以访问该属性值。即其删除的是字典对象，其实例对象不会删除。

举例说明

如何使用上下文。

第一步 创建 package：

```
CREATE or replace package test_pk as
    procedure set_context(ts_name varchar, key varchar, value varchar);
    procedure set_user_context(ts_name varchar, key varchar, value varchar, username varchar, client_id varchar);
    procedure get_context(ts_name varchar, key varchar);
```

```
    procedure clear_context(ts_name varchar, key varchar, value varchar);
    end test_pk;
    /
    CREATE or replace package body test_pk as
        procedure set_context(ts_name varchar, key varchar, value varchar) as
        begin
            dbms_session.set_context(ts_name, key, value);
        end;

    procedure set_user_context(ts_name varchar, key varchar, value varchar,
username varchar, client_id varchar) as
    begin
     dbms_session.set_context(ts_name, key, value, username, client_id);
    end;

    procedure get_context(ts_name varchar, key varchar) as
    begin
        dbms_output.put_line('==' || sys_context(ts_name, key) || '--');
    end;

    procedure clear_context(ts_name varchar, key varchar, value varchar) as
    begin
        dbms_session.clear_context(ts_name, key, value);
    end;
    end test_pk;
    /
```

第二步 创建 context

```
create or replace context c_user01 using test_pk;
```

第三步 设置 namespace 的属性

```
call test_pk.set_context('c_user01', 'u_k2', 'u_v2');
```

第四步 查询该属性值

```
call test_pk.get_context('c_user01', 'u_k2');
```

打印值为：

```
==u_v2--
```

3.15 管理目录

3.15.1 创建目录

创建一个目录对象。目录是操作系统文件在数据库中的一个映射。

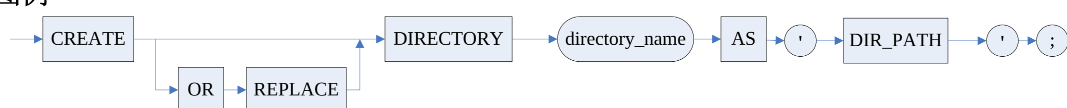
语法格式

```
CREATE [OR REPLACE] DIRECTORY <目录名> AS 'DIR_PATH';
```

参数

1. <目录名> 创建的目录的名称

图例



语句功能

供具有 CREATE ANY DIRECTORY 权限的用户创建一个目录对象。

3.15.2 删除目录

删除一个目录对象。

语法格式

```
DROP DIRECTORY  <目录名>;
```

参数

1. <目录名> 要删除的目录的名称

图例



语句功能

供具有 DROP ANY DIRECTORY 权限的用户删除一个目录对象。

3.16 设置当前会话

3.16.1 时区信息

设置当前会话时区信息

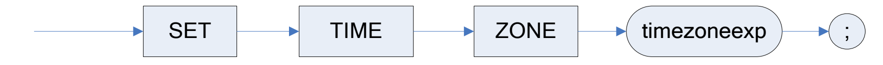
语法格式

```
SET TIME ZONE <时区>;
<时区>::= <LOCAL>|<’[+|-]整数’>|INTERVAL‘[+|-]整数’<间隔类型>
```

参数

<时区> 指明要设置的时区信息；

图例



语句功能

设置当前会话时区信息。

使用说明

仅当前会话有效。

举例说明

例 设置当前会话时区为‘+9:00’。

```
SET TIME ZONE  '+9:00';
```

例 设置当前会话时区为服务器所在地时区。

```
SET TIME ZONE LOCAL;
```

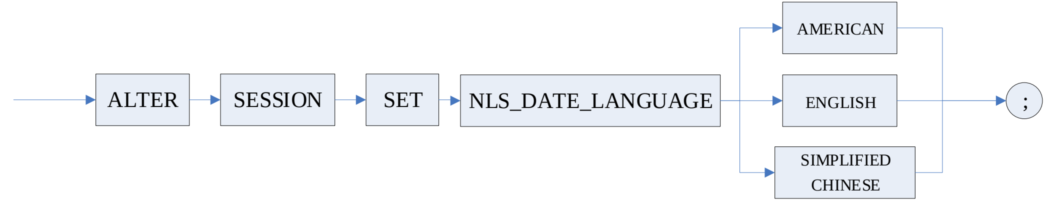
3.16.2 日期串语言

设置当前会话的日期串语言。

语法格式

```
ALTER SESSION SET NLS_DATE_LANGUAGE=<语言>;
<语言>:=[AMERICAN]|[ENGLISH]|[SIMPLIFIED CHINESE]
```

图例



语句功能

设置当前会话日期串语言。

使用说明

仅当前会话有效。

举例说明

例 设置当前会话日期串为 ENGLISH。

```
ALTER SESSION SET NLS_DATE_LANGUAGE=ENGLISH;
```

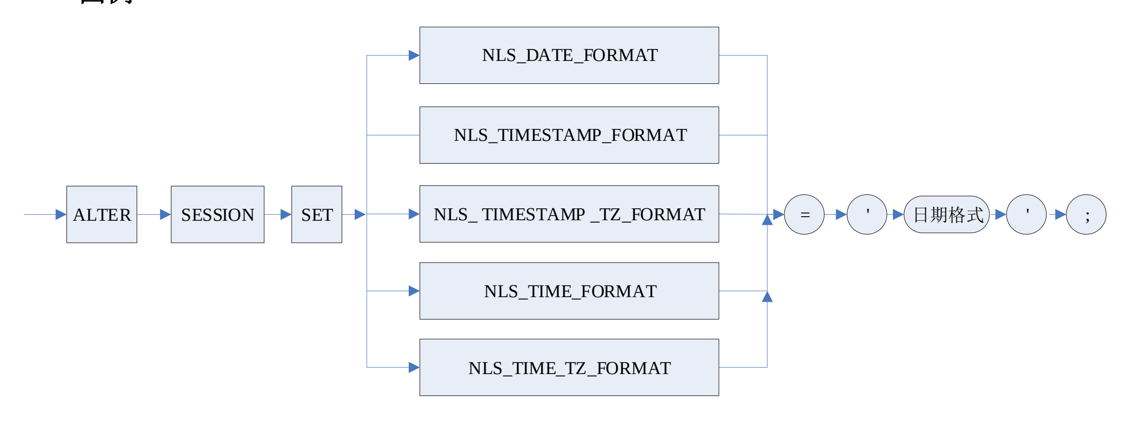
3.16.3 日期串格式

设置当前会话的日期串格式。

语法格式

```
ALTER SESSION SET <NLS_DATE_FORMAT | NLS_TIMESTAMP_FORMAT | NLS_ TIMESTAMP
_TZ_FORMAT | NLS_TIME_FORMAT | NLS_TIME_TZ_FORMAT> = <日期格式>;
```

图例



语句功能

设置当前会话的日期串格式。

参数

1. NLS_DATE_FORMAT/ NLS_TIMESTAMP_FORMAT / NLS_ TIMESTAMP _TZ_FORMAT / NLS_TIME_FORMAT / NLS_TIME_TZ_FORMAT：目前这几个关键词作用都是一样的；

2. <日期格式> 有三种格式：DATE 格式、TIME 格式或 DATE 格式+TIME 格式三种格式。其中，DATE 格式为年月日、月日年或日月年，各部分之间可以有间隔符(".","-","/"等分隔符)连接或者没有间隔符；TIME 格式为：时分或时分秒，间隔符为":"。例如，"YYYYMMDD HH:MI:SS"、"YYYY.MM.DD HH:MI:SS"、"YYYY-MM-DD HH:MI:SS"或是"YYYY/MM/DD HH:MI:SS"等。

使用说明

仅当前会话有效。

举例说明

例 设置当前会话日期格式为 YYYY-MM-DD。

```
ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD';
```

3.17 注释语句

可以通过注释语句来创建或修改表、视图或它们的列的注释信息。表和视图上的注释信息可以通过查询字典表 SYSTABLECOMMENTS 进行查看，列的注释信息可以通过查询字典表 SYSCOLUMNCOMMENTS 进行查看。

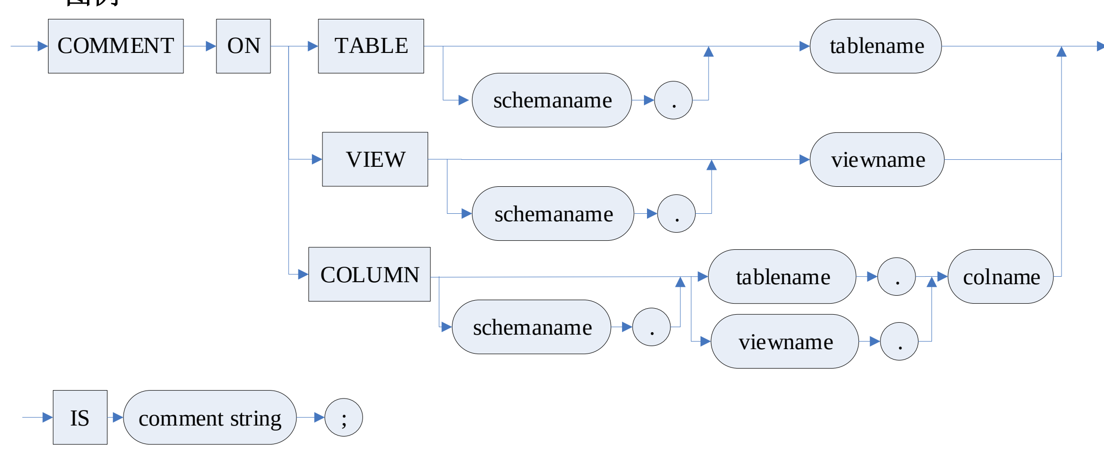
语法格式

```
COMMENT ON  <对象名称> IS <注释字符串>
<对象名称> ::=
    TABLE <表名定义> |
    VIEW <视图名定义> |
    COLUMN <列名定义>
<表名定义> ::= [<模式名>.]<表名>
<视图名定义> ::= [<模式名>.]<视图名>
<列名定义> ::=
    [<模式名>.]<表名>.<列名>|
    [<模式名>.]<视图名>.<列名>
```

参数

<注释字符串> 指明要设置的注释信息；

图例



语句功能

供具有 DBA 或 COMMENT ANY TABLE 权限的用户为表、视图或列创建注释信息。

使用说明

1．用户只能为自己所拥有模式中的表、视图和列对象创建注释信息；

2．注释字符串最大长度为4000；

3．在已有注释的对象上再次执行此语句将直接覆盖之前的注释。

举例说明

例 1 为表 PERSON 创建注释信息。

```
COMMENT ON TABLE PERSON IS 'PERSON IS A SIMPLE TABLE';
```

例 2 为表 PERSON 的列 NAME 创建注释信息。

```
COMMENT ON COLUMN PERSON.NAME IS 'SYSDBA.PERSON.NAME';
```

3.18 设置 INI 参数

INI 参数分为手动、静态和动态三种类型，分别对应 V$PARAMETER 视图中 TYPE 列的 READ ONLY、IN FILE、SYS/SESSION。服务器运行过程中，手动（READ ONLY）参数不能被修改，静态和动态参数可以修改。

静态（IN FILE）参数只能通过修改 dm.ini 文件进行修改，修改后重启服务器才能生效，为系统级参数，生效后会影响所有的会话。

动态（SYS 和 SESSION）参数可在 dm.ini 文件和内存同时修改，修改后即时生效。其中，SYS 为系统级参数，修改后会影响所有的会话；SESSION 为会话级参数，服务器运行过程中被修改时，之前创建的会话不受影响，只有新创建的会话使用新的参数值。

3.18.1 设置参数值

用户可以通过ALTER SYSTEM语法修改静态或动态（系统级、会话级）参数值，使修改之后的参数值能够在全局范围内起作用。对于静态参数，只有指定SPFILE情况下，才能修改。

语法格式

```
ALTER SYSTEM SET '<参数名称>' =<参数值> [DEFERRED] [MEMORY|BOTH|SPFILE];
```

参数

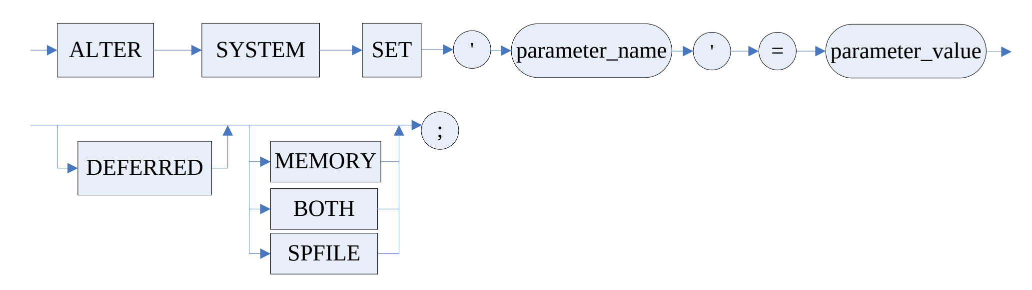
1．<参数名称> 指静态、动态（系统级、会话级）INI参数名字；

2．<参数值> 指设置该INI参数的值；

3．[DEFERRED] 只适用于动态参数。指定DEFERRED，参数值延迟生效，对当前会话不生效，只对新创建的会话生效；缺省为立即生效，对当前会话和新创建的会话都生效；

4．[MEMORY|BOTH|SPFILE] 设置INI参数修改的位置。其中，MEMORY只对内存中的INI值做修改；SPFILE则只对INI文件中的INI值做修改；BOTH则内存和INI文件都做修改。默认情况下，为MEMORY。对于静态参数，只能指定SPFILE。

图例



语句功能

设置系统级的 INI 参数值。

举例说明

例 设置当前系统动态、会话级参数 SORT_BUF_SIZE 参数值为 200，要求延迟生效，对当前的 session 不生效，对后面创建的会话才生效。并且只修改内存。

```
ALTER SYSTEM SET 'SORT_BUF_SIZE' =200  DEFERRED  MEMORY;
```

例 设置静态参数 MTAB_MEM_SIZE 参数值为 1200。

```
ALTER SYSTEM SET 'MTAB_MEM_SIZE' =1200  spfile;
```

3.18.2 设置仅对当前会话起作用

用户可以通过 ALTER SESSION 语法修改动态会话级参数（即 TYPE 为 SESSION 的参数），使修改之后的 INI 参数值只对当前会话起作用，不会影响其他会话或系统的 INI 参数值。

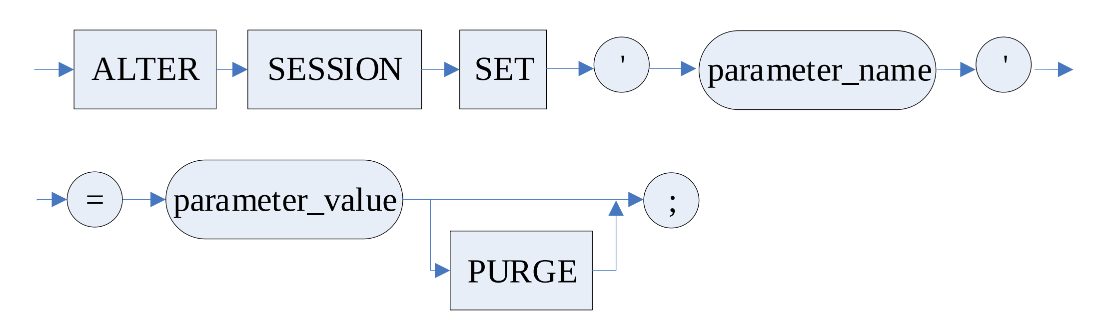
语法格式

```
ALTER SESSION SET '<参数名称>' =<参数值> [PURGE];
```

参数

1. <参数名称> 指动态会话级INI参数名字；
2. <参数值> 指设置该INI参数的相应值；
3. [PURGE] 指是否清理执行计划。

图例



语句功能

设置动态、会话级的 INI 参数值。

使用说明

设置后的值只对当前会话有效。当包含 PURGE 选项时会清除服务器保存的所有执行计划。

举例说明

例 设置当前会话的 HAGR_HASH_SIZE 参数值为 2000000。

```
ALTER SESSION SET 'HAGR_HASH_SIZE' =2000000;
```

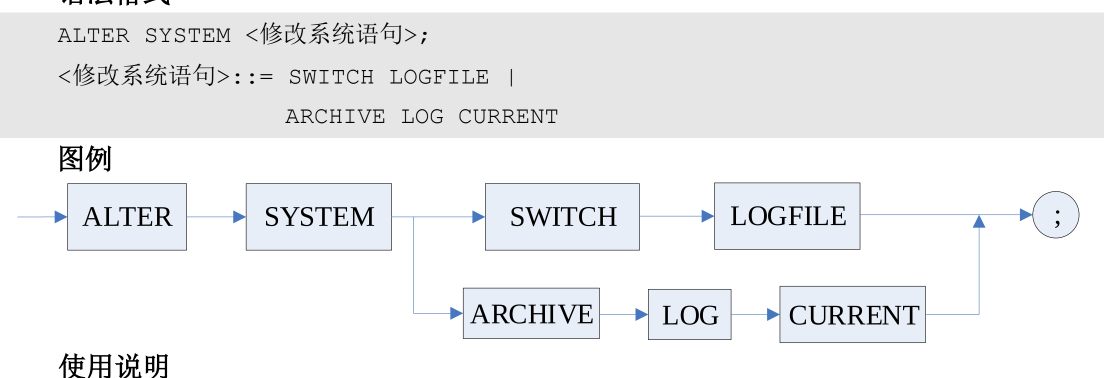
3.19 修改系统语句

用户可以通过 ALTER SYSTEM 语句对系统进行修改，如设置系统级别参数（参见 3.18.1 设置参数值），也可以使用系统修改语句切换归档文件和归档当前所有的 REDO 日志。

语法格式

```
ALTER SYSTEM <修改系统语句>;
<修改系统语句>::= SWITCH LOGFILE |
                  ARCHIVE LOG CURRENT
```

图例



使用说明

ARCHIVE LOG CURRENT 和 SWITCH LOGFILE 功能一样，都是把新生成的，还未归档的联机日志进行归档。

3.20 设置列、索引生成统计信息

设置列、索引生成统计信息。

语法格式

```
STAT <统计信息采样率百分比> [SIZE <直方图桶数>] ON <统计对象> [GLOBAL]
<统计对象>:: =
[<模式名>.] <表名> (<列名>{,<列名>}) |
INDEX [<模式名>.]<索引名>
```

参数

1. <统计信息采样率百分比> 指定统计信息采样率的百分比。必须为[0,100]范围内整数；

2. <直方图桶数> 指定统计信息的直方图桶数，单列取值范围为0或[1~254]范围内的整数，其中0表示不限制。不指定时系统根据数据的实际情况动态确定。多列的取值范围是[1,2500]；

3. <模式名> 指定收集统计信息的模式。缺省为当前会话的模式名；

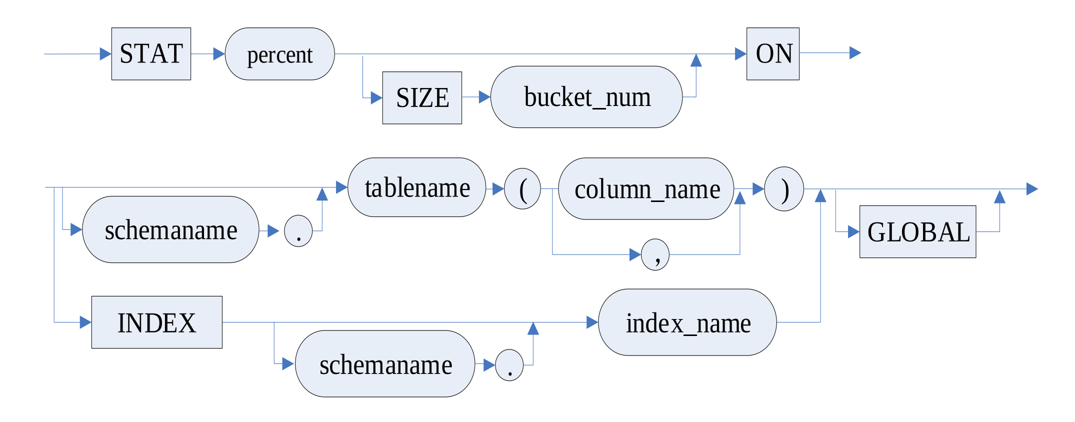
4. <表名> 指定收集统计信息的表；

5. <列名> 指定收集统计信息的列。当前最大支持127列；

6. <索引名> 指定收集统计信息的索引；

7. GLOBAL 用于MPP环境下各节点数据收集后统一生成统计信息。

图例



语句功能

为列或索引生成的统计信息。

使用说明

1． 不支持所在表空间为OFFLINE的对象；

2． <表名>不支持外部表、DBLINK远程表、动态视图表、记录类型数组所用的临时表；

3． <列名>不支持 ROWID、ROWNUM等特殊列，不支持BLOB、IMAGE、LONGVARBINARY、CLOB、TEXT、LONGVARCHAR、自定义类型列和空间类型列等列类型；

4． <索引名>，不支持位图索引、位图连接索引、虚索引、全文索引、空间索引、数组索引、无效的索引；

5． 该语句的调用将导致当前事务被提交；

6． GLOBAL 仅在MPP环境下使用GLOBAL登录时可用，否则报错；

7． 多列统计信息的只支持表，不支持索引。

举例说明

例 1 对 SYSOBJECTS 表上 ID 列生成统计信息，采样率的百分比为 30%。

```
STAT 30 ON SYS.SYSOBJECTS (ID);
```

例 2 对 PURCHASING 模式下的索引 S1 生成统计信息，采样率为 50%。

```
STAT 50 ON INDEX PURCHASING.S1;
```

例 3 对 SYSOBJECTS 表上 PID,NAME 列生成统计信息，采样率的百分比为 30%。

```
STAT 30 ON SYS.SYSOBJECTS (PID,NAME);
```

3.21设置表生成统计信息

设置表生成统计信息。

语法格式

```
STAT  ON  [<模式名>.]<表名> [GLOBAL];
```

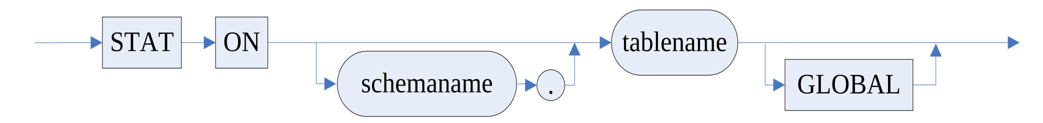
参数

1． <模式名> 指定生成统计信息的表的模式。缺省为当前会话的模式名；

2． <表名> 指定生成统计信息的表；

3． GLOBAL 用于MPP环境下各节点数据收集后统一生成统计信息。

图例



语句功能

为表生成统计信息。

使用说明

1. 不支持为所在表空间为OFFLINE的表生成统计信息；
2. 不支持为外部表、DBLINK远程表、动态视图表、记录类型数组所用的临时表生成统计信息；
3. 该语句的调用将导致当前事务被提交；
4. GLOBAL 仅在MPP环境下使用GLOBAL登录时可用，否则报错。

举例说明

例 1 为 SYSOBJECTS 表生成统计信息。

```
STAT ON SYS.SYSOBJECTS;
```

第 4 章 数据查询语句

数据查询是数据库的核心操作，DM_SQL 语言提供了功能丰富的查询方式，满足实际应用需求。几乎所有的数据库操作均涉及到查询，因此熟练掌握查询语句的使用是数据库从业人员必须掌握的技能。

在 DM_SQL 语言中，有的定义语法中也包含查询语句，如视图定义语句、游标定义语句等。为了区别，我们将这类出现在其它定义语句中的查询语句称查询说明。

每种查询都有适用的场景，使用得当会大大提高查询效率。为方便用户的使用，本章对 DM_SQL 语言支持的查询方式进行讲解，测例中所用基表及各基表中预先装入的数据参见第 2 章，各例的建表者均为用户 SYSDBA。

查询语句的语法如下：

```
<查询表达式>::=
    <simple_select>|
    <select_clause> <ORDER BY 子句> [<FOR UPDATE 子句>] [<LIMIT 限制条件>]|
    <select_clause> <LIMIT 子句> <ORDER BY 子句> [<FOR UPDATE 子句>] |
    <select_clause> <FOR UPDATE 子句> [<LIMIT 限制条件>] |
    <select_clause> <LIMIT 限制条件>[<FOR UPDATE 子句>]
<simple_select> ::=
    <query_exp_with>|
    <select_clause> <UNION| EXCEPT | MINUS | INTERSECT > [ALL | DISTINCT | UNIQUE] [CORRESPONDING [BY (<列名> {,<列名>})]] <select_clause>
<select_clause>::=
    <simple_select>|
    (<查询表达式>)|
    (<select_clause>)
<ORDER BY 子句> ::= ORDER [SIBLINGS] BY <order_by_list> [ASC | DESC] [NULLS FIRST|LAST]{,< order_by_list> [ASC | DESC] [NULLS FIRST|LAST]}
<order_by_list>::=
    <无符号整数> |
    <列说明> |
    <值表达式>
<FOR UPDATE 子句> ::=
    FOR READ ONLY|
    FOR UPDATE [OF <选择列表>] [NOWAIT | WAIT N | SKIP LOCKED]
<LIMIT 限制条件> ::=<LIMIT 子句>|< ROW_LIMIT 子句>
<LIMIT 子句>::=LIMIT< <记录数> | <<记录数>,<记录数>>| <<记录数> OFFSET <偏移量>> >
<记录数>::=<整数>
<偏移量>::=<整数>
< ROW_LIMIT 子句>::= [OFFSET <offset> <ROW | ROWS> ] [<FETCH 说明>]
<FETCH 说明>::= FETCH <FIRST | NEXT> <大小> [PERCENT] < ROW | ROWS > <ONLY| WITH TIES>
<query_exp_with>::=[<WITH 子句>] SELECT [<HINT 子句>] [ALL | DISTINCT |
```

```
UNIQUE] [<TOP 子句>] <选择列表>[<bulk_or_single_into_null>] <select_tail>
    <选择列表> ::= [[<模式名>.]<基表名> | <视图名> .] * | <值表达式> [[AS] <列别名>]
                {,[[<模式名>.]<基表名> | <视图名>.] * | <值表达式> [[AS] <列别名>]}
    <WITH 子句> ::= [<WITH FUNCTION 子句>] [WITH CTE 子句]   请参考第 4.4 节 WITH 子句
    <HINT 子句> ::=/*+ hint{hint}*/
     <TOP 子句>::=
        TOP <n> |
        <<n>,<n>>|
        <n> PERCENT|
        <n> WITH TIES|
        <n> PERCENT WITH TIES
    <n>::=整数（>=0）
    <bulk_or_single_into_null>::=<bulk_or_single_into>  <变量名 >{,<变量名>}
    <bulk_or_single_into>::= <INTO>| <BULK COLLECT INTO>
    <select_tail>::=
     <FROM 子句>
     [<WHERE 子句>]
     [<层次查询子句>]
     [<GROUP BY 子句>]
     [<HAVING 子句>]
    <FROM 子句>::= FROM <表引用>{,<表引用>}
    <表引用>::=<普通表>|<连接表>
    <普通表>::=<普通表 1>|<普通表 2>|<普通表 3>|<ARRAY<数组>>
    <普通表 1>::=<对象名> [<SAMPLE 子句>][[AS <别名>] <PIVOT 子句>][[AS <别名>]
<UNPIVOT 子句>] [<闪回查询>] [[AS] <别名>]
    <普通表 2>::=(<查询表达式>)[[AS <别名>] <PIVOT 子句>][[AS <别名>] <UNPIVOT 子句>]
[<闪回查询>][[AS] <表别名> [<新生列>]]
    <普通表 3>::=[<模式名>.]<<基表名>|<视图名>>(<选择列>)[[AS <别名>] <PIVOT 子句>]
[[AS <别名>] <UNPIVOT 子句>] [<闪回查询>] [[AS] <表别名> [<派生列表>]]
    <对象名>::=<本地对象> | <索引> | <分区表>
    <本地对象>::=[<模式名>.]<基表名|视图名>
    <索引>::=[<模式名>.]<基表名> INDEX <索引名>
    <分区表>::=
        [<模式名>.]<基表名> PARTITION (<一级分区名>) |
        [<模式名>.]<基表名> SUBPARTITION (<子分区名>)
    <选择列>::=<列名>[{,<列名> }]
    <派生列表>::=(<列名>[{,<列名>}])
    <SAMPLE 子句>::=
        SAMPLE(<表达式>) |
        SAMPLE(<表达式>) SEED (<表达式>) |
        SAMPLE BLOCK(<表达式>)  |
        SAMPLE BLOCK(<表达式>) SEED (<表达式>)
    <闪回查询>::= <闪回查询子句>|<闪回版本查询子句>请参考第 18 章 闪回查询
    <闪回查询子句>::=WHEN <TIMESTAMP time_exp> | <TRXID trxid>
    <闪回版本查询子句>::=VERSIONS BETWEEN <TIMESTAMP time_exp1 AND time_exp2> |
<TRXID trxid1 AND trxid2>
     <连接表>::=[(]<交叉连接>|<限定连接>[)]
```

```
    <交叉连接>::=<表引用> CROSS JOIN <<普通表>|(<连接表>)>
    <限定连接>::=<表引用> [<PARTITION BY 子句>] [NATURAL] [<连接类型>] JOIN <<普通表>|(<连接表>)> [<PARTITION BY 子句>]
    <连接类型>::=
         [<内外连接类型>]<INNER>|
         <内外连接类型>[<OUTER>]
    <内外连接类型>::=LEFT|RIGHT|FULL
    <连接条件>::=<条件匹配>|<列匹配>
    <条件匹配>::=ON<搜索条件>
    <列匹配>::=USING(<连接列列名>{, <连接列列名>})
    <WHERE 子句> :: =
         WHERE <搜索条件>|
         < WHERE CURRENT OF 子句>
    <搜索条件>::=<逻辑表达式>
    < WHERE CURRENT OF 子句>:: =WHERE CURRENT OF <游标名>
    <层次查询子句>::=
         CONNECT BY [NOCYCLE] <连接条件>[START WITH <起始条件> ] |
         START WITH <起始条件> CONNECT BY [NOCYCLE] <连接条件>
    <连接条件>::=<逻辑表达式>
    <起始条件>::=<逻辑表达式>
    <GROUP BY 子句> ::= GROUP BY <group_by 项>{,<group_by 项>}
    <group_by 项>::=<分组项>|<ROLLUP 项>|<CUBE 项>|<GROUPING SETS 项>
    <分组项>::=<值表达式>
    <ROLLUP 项>::=ROLLUP (<分组项>)
    <CUBE 项>::=CUBE (<分组项>)
    <GROUPING SETS 项>::=GROUPING SETS(<GROUP 项>{,<GROUP 项>})
    <GROUP 项>::=
         <分组项> |
         (<分组项>{,<分组项>})|
         ()
    <HAVING 子句> ::= HAVING <搜索条件>
    <PARTITION BY 子句> ::=PARTITION BY (<表列名>{,<表列名>})
    <PIVOT 子句> ::= PIVOT [XML] ( (<集函数> {,<集函数>}) <pivot_for_clause> IN (<pivot_in_clause>) )
    <pivot_for_clause> ::=
         FOR <列名> |
         FOR (<列名>{,<列名>})
    <pivot_in_clause> ::= <表达式> [ [AS]  <别名>] {, <表达式> [[AS]  <别名>]} |
                          (<表达式>) [ [AS]  <别名>] {, (<表达式>) [[AS]  <别名>]} |
                          <select_clause> |
                          ANY
    <UNPIVOT 子句> ::= UNPIVOT [<include_null_clause>](<unpivot_val_col_lst>
    <pivot_for_clause> IN (<unpivot_in_clause_low> ))
    <include_null_clause> ::=
```

```
        INCLUDE NULLS |
        EXCLUDE NULLS
    <unpivot_val_col_lst> ::=
        <表达式> |
        (<表达式>{,<表达式>})
    <unpivot_in_clause_low> ::= <unpivot_in_clause>{, <unpivot_in_clause>}
    <unpivot_in_clause> ::=
        <列名> [AS <别名>] |
        (<列名>{,<列名>}) [ AS (<别名>{,<别名>})] |
        (<列名>{,<列名>}) AS <别名>
```

参数

1. ALL　　返回所有被选择的行，包括所有重复的拷贝，缺省值为 ALL；

2. DISTINCT　从被选择出的具有重复行的每一组中仅返回一个这些行的拷贝，与 UNIQUE 等价。对于集合算符：UNION，缺省值为 DISTINCT，DISTINCT 与 UNIQUE 等价；对于 EXCEPT/MINUS 和 INTERSECT：操作的两个表中数据类型和个数要完全一致。其中，EXCEPT 和 MINUS 集合算符功能完全一样，返回两个集合的差集；INTERSECT 返回两个集合的交集（去除重复记录）；

3. CORRESPONDING 用于指定列名链表，通过指定列名（或列名的别名）链表来对两个查询分支的查询项进行筛选。无论分支中有多少列，最终的结果集只包含 CORRESPONDING 指定的列。查询分支和 CORRESPONDING 的关系为：<查询分支 1> CORRESPONDING [BY (<列名> {,<列名>})] <查询分支 2>。如果 CORRESPONDING 指定了列名但两个分支中没有相同列名的查询项则报错，如果 CORRESPONDING 没指定列名，则按照第一个分支的查询项列名进行筛选；例如：select c1, c2, c3 from t1 union all corresponding by (c1,c2) select d1, d2 c1, d3 c2 from t2;

4. hint　　用于优化器提示，可以出现在语句中任意位置，具体可使用的 hint 可通过 V$HINT_INI_INFO 动态视图查询；

5. <模式名>　　被选择的表和视图所属的模式，缺省为当前模式；

6. <基表名>　　被选择数据的基表的名称；

7. <视图名>　　被选择数据的视图的名称；

8. *　　指定对象的所有的列；

9. <值表达式>　可以为一个<列引用>、<集函数>、<函数>、<标量子查询>或<计算表达式>等等；

10. <列别名>　　为列表达式提供不同的名称，使之成为列的标题，列别名不会影响实际的名称，别名在该查询中被引用；

11. <相关名>　　给表、视图提供不同的名字，经常用于求子查询和相关查询的目的；

12. <列名>　　指明列的名称；

13. <WHERE 子句>　限制被查询的行必须满足条件，如果忽略该子句，DM 从在 FROM 子句中的表、视图中选取所有的行；其中，<WHERE CURRENT OF 子句>专门用于游标更新、删除中，用来限定更新、删除与游标有关的数据行。

14. <HAVING 子句>　限制所选择的行组所必须满足的条件，缺省为恒真，即对所有的组都满足该条件；

15. <无符号整数>　　指明了要排序的<值表达式>在 SELECT 后的序列号；

16. <列说明>　　排序列的名称；

17. ORDER SIBLINGS BY 必须与 CONNECT BY 一起配合使用。可用于指定层次查询中相同层次数据返回的顺序。

18. ASC 指明为升序排列，缺省为升序；

19. DESC 指明为降序排列；

20. NULLS FIRST 指定排序列的 NULL 值放在最前面，不受 ASC 和 DESC 的影响，缺省的是 NULLS FIRST；

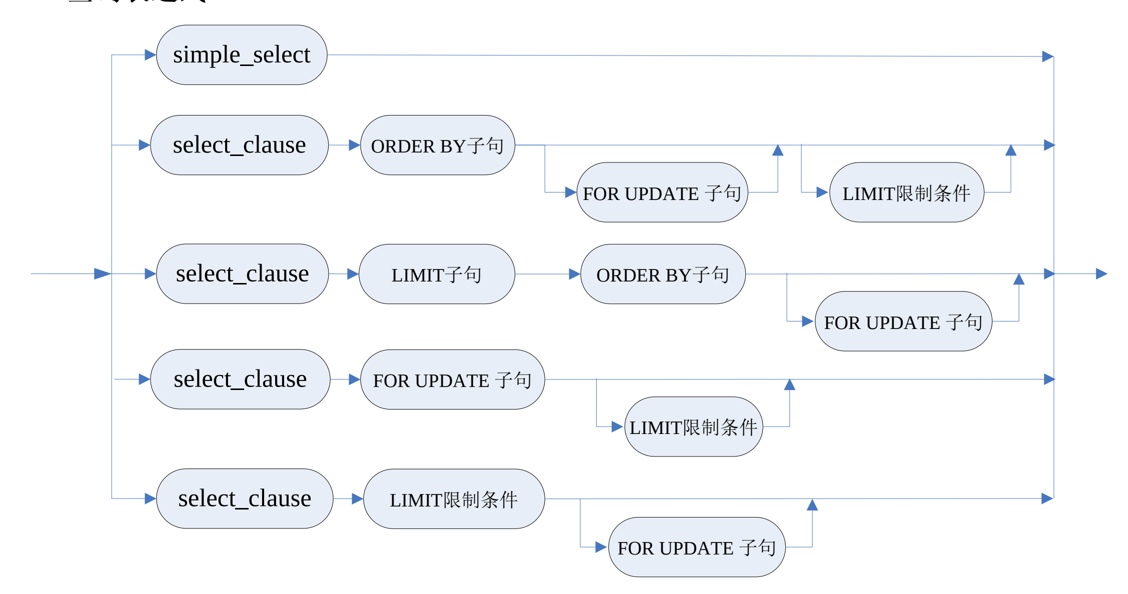
21. NULLS LAST 指定排序列的 NULL 值放在最后面，不受 ASC 和 DESC 的影响；

22. <PARTITION BY 子句> 指明分区外连接中的分区项，最多支持 255 个列；仅允许出现在左外连接右侧表和右外连接中的左侧表，且不允许同时出现，详见 4.2.7；

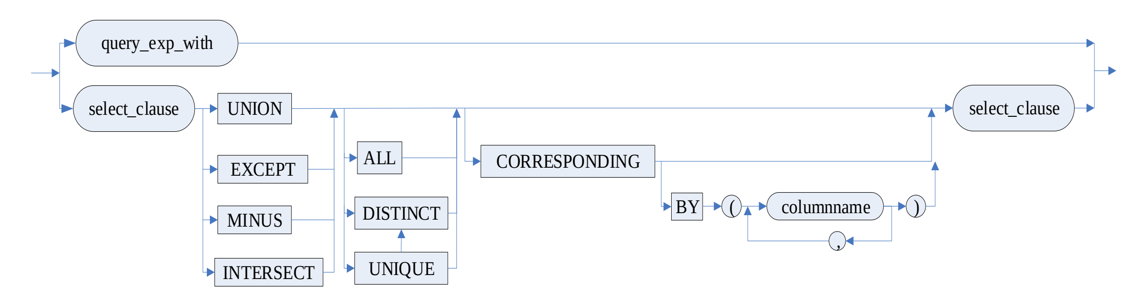
23. BULK COLLECT INTO 的作用是将检索结果批量的、一次性的赋给集合变量。与每次获取一条数据，并每次都要将结果赋值给一个变量相比，可以很大程度上的节省开销。使用 BULK COLLECT 后，INTO 后的变量必须是集合类型。

图例

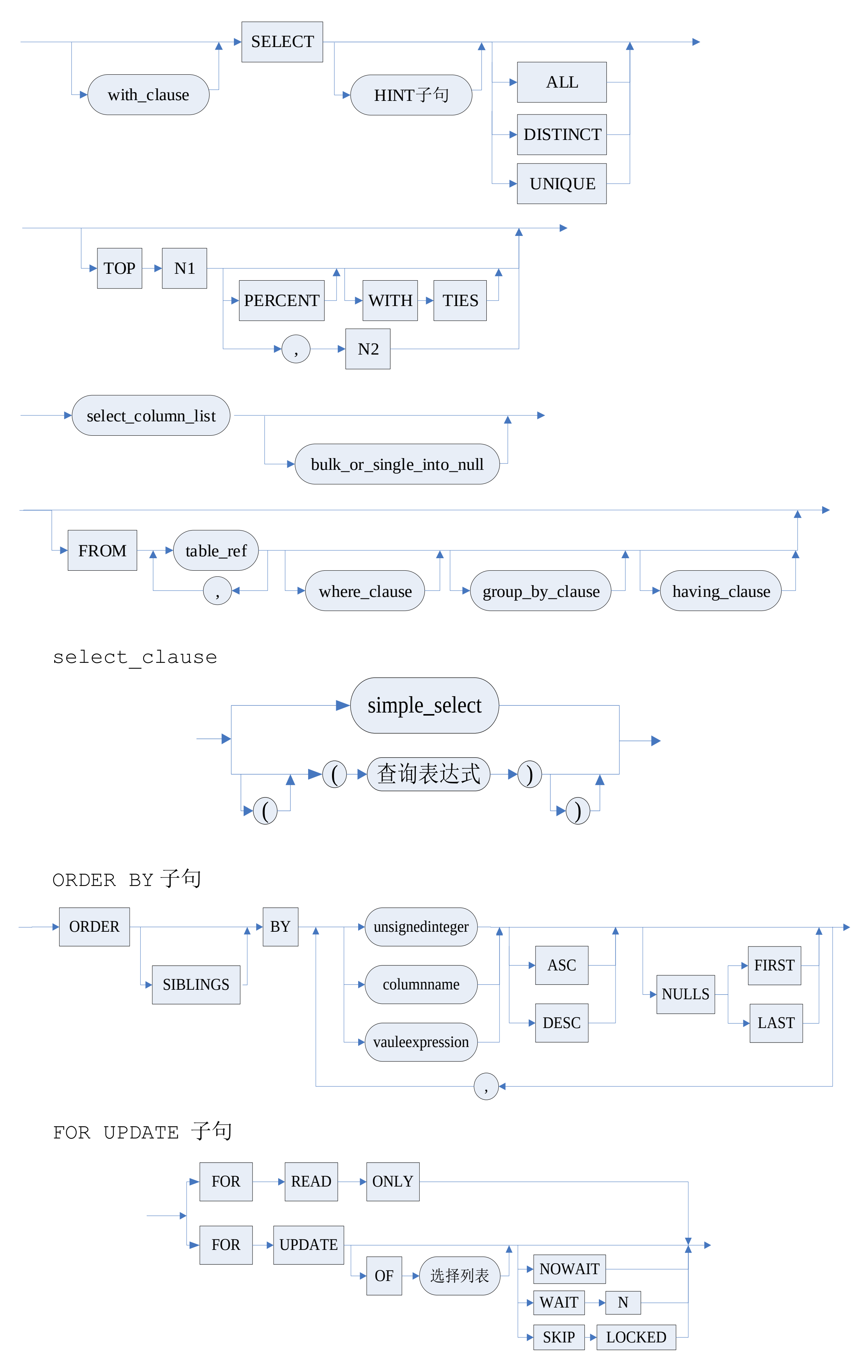
查询表达式



simple_select

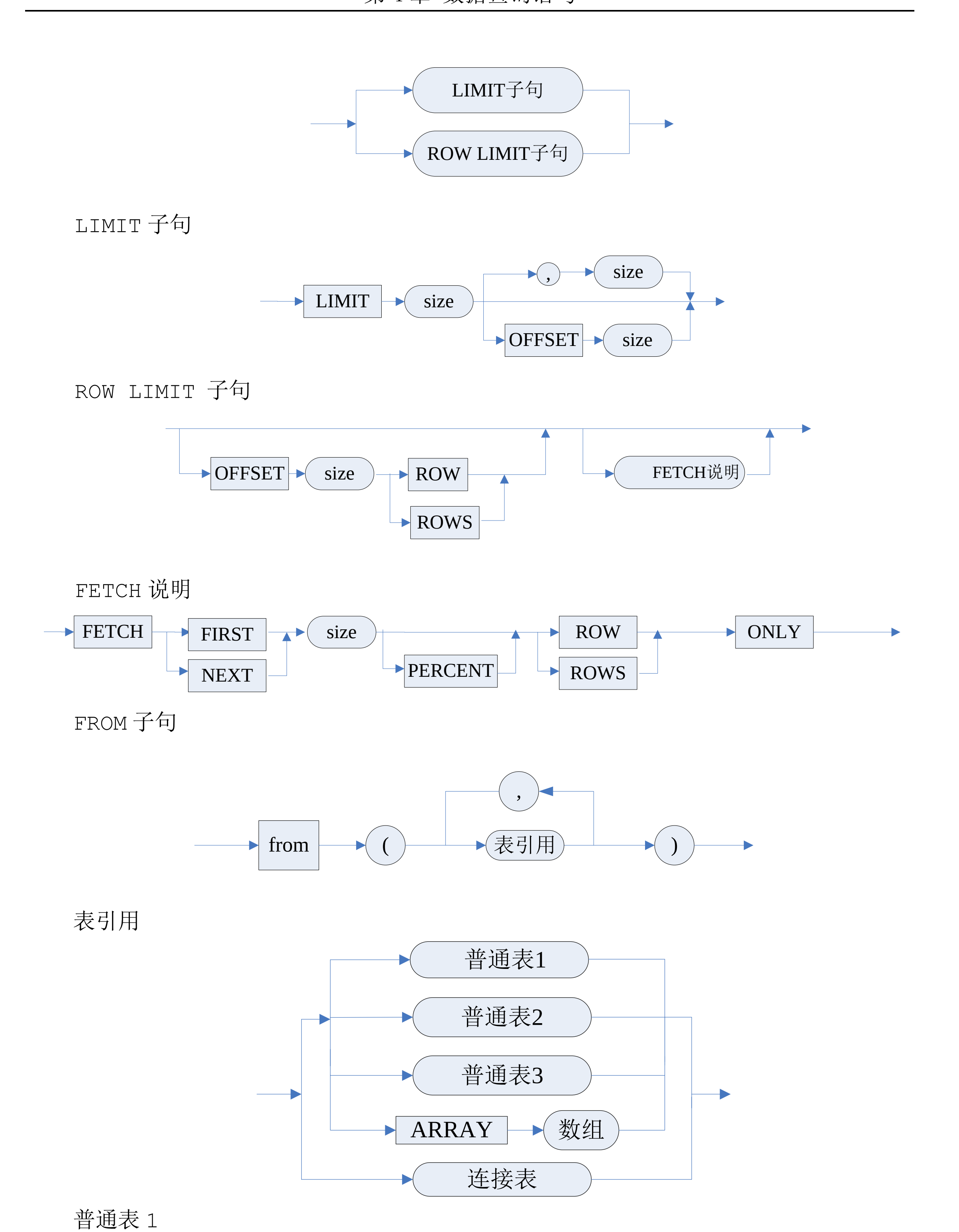


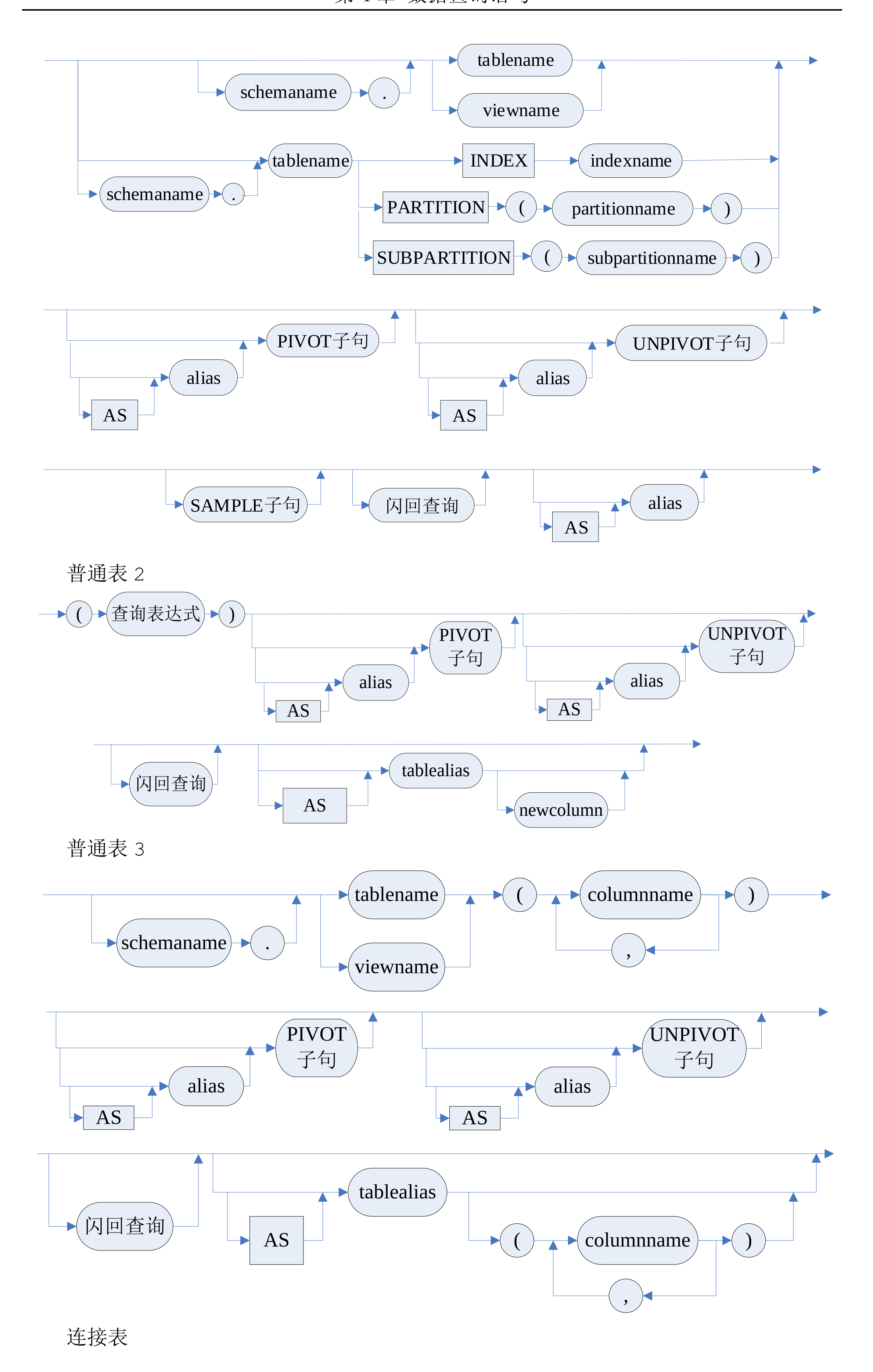
query_exp_with

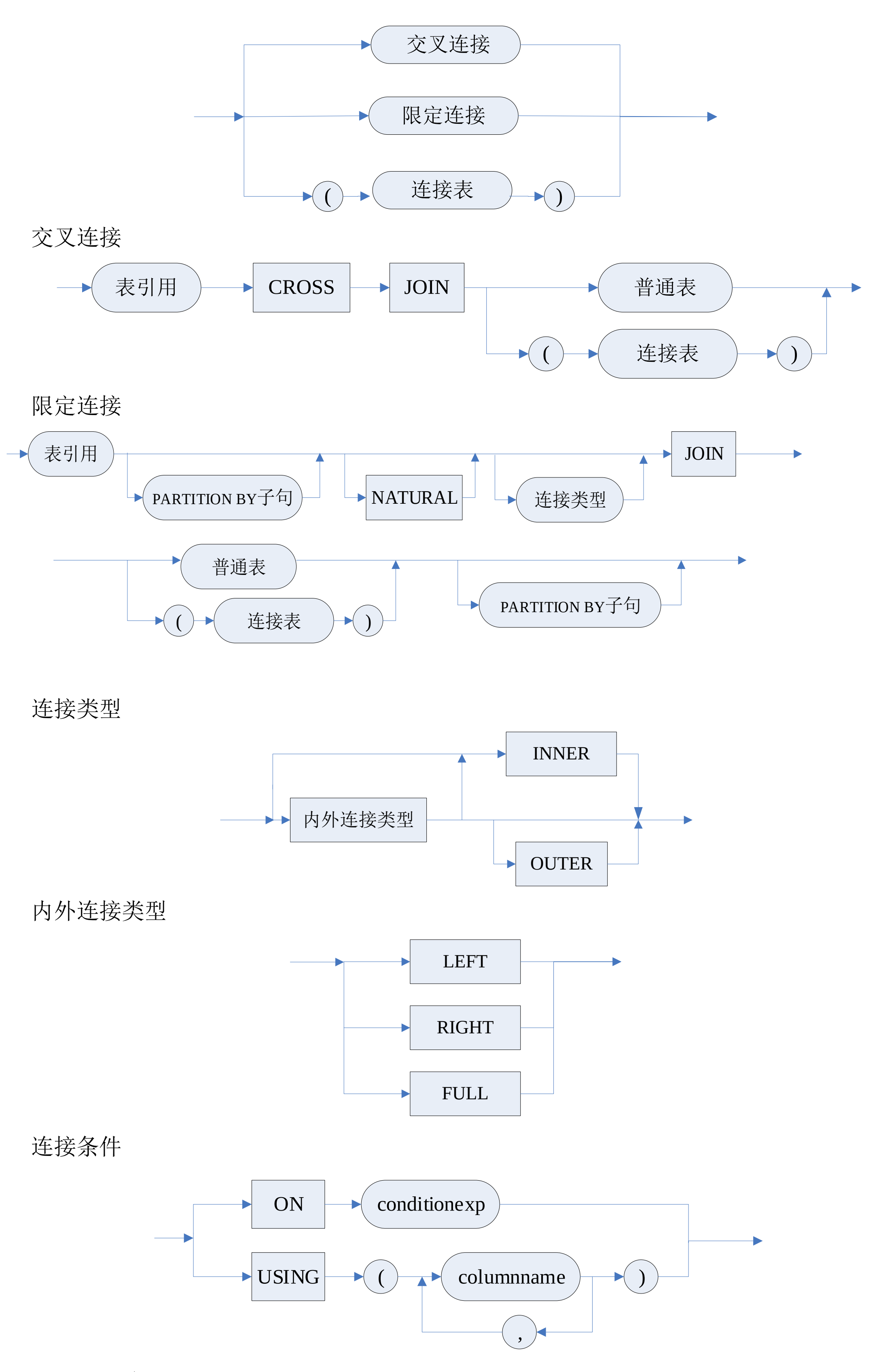


SELECT
with_clause
HINT子句
ALL
DISTINCT
UNIQUE
TOP
N1
PERCENT
WITH
TIES
,
N2
select_column_list
bulk_or_single_into_null
FROM
table_ref
,
where_clause
group_by_clause
having_clause
select_clause
simple_select
(
查询表达式
)
(
)
ORDER BY 子句
ORDER
SIBLINGS
BY
unsignedinteger
columnname
vauleexpression
ASC
DESC
NULLS
FIRST
LAST
,
FOR UPDATE 子句
FOR
READ
ONLY
FOR
UPDATE
OF
选择列表
NOWAIT
WAIT
N
SKIP
LOCKED

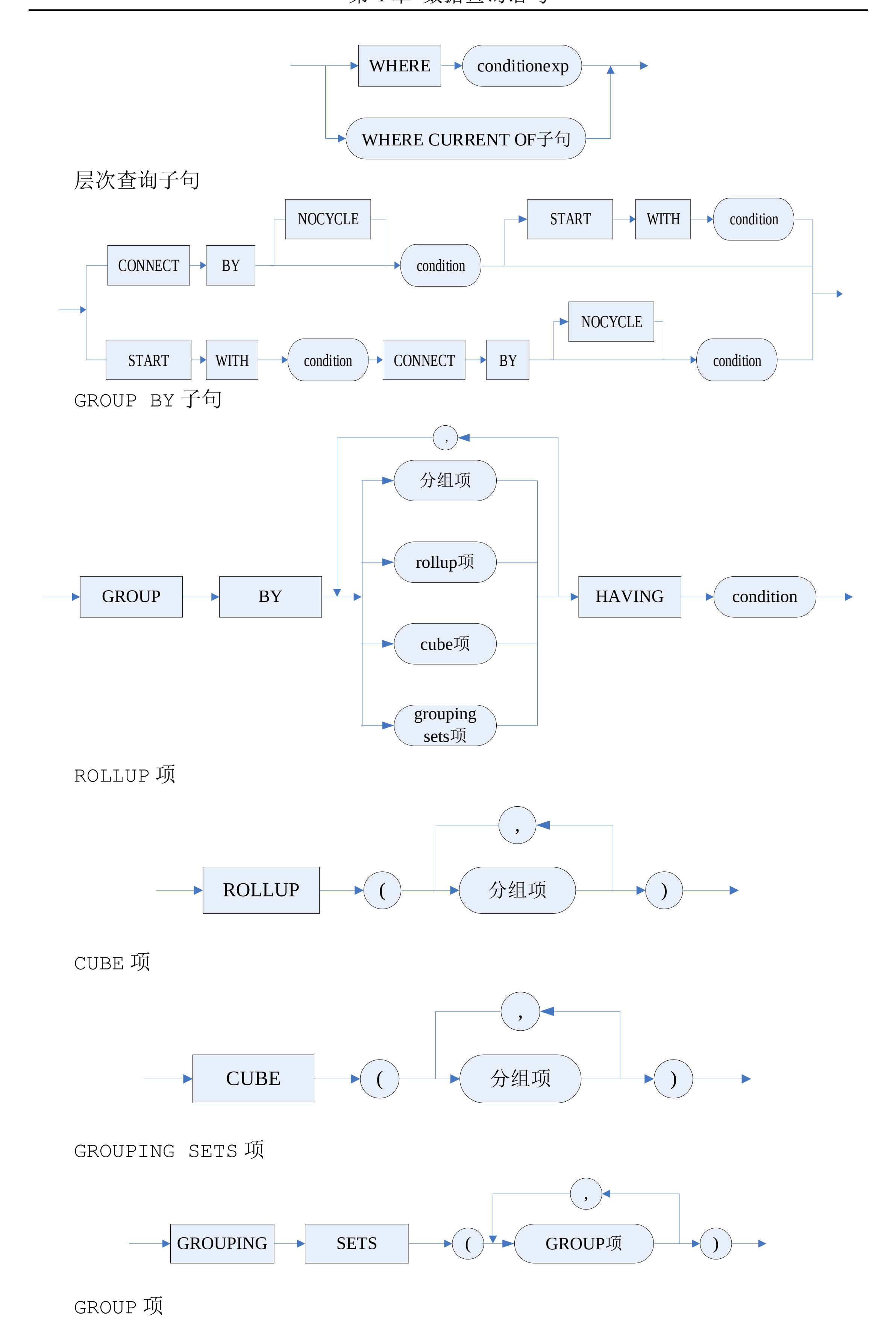
LIMIT 限制条件

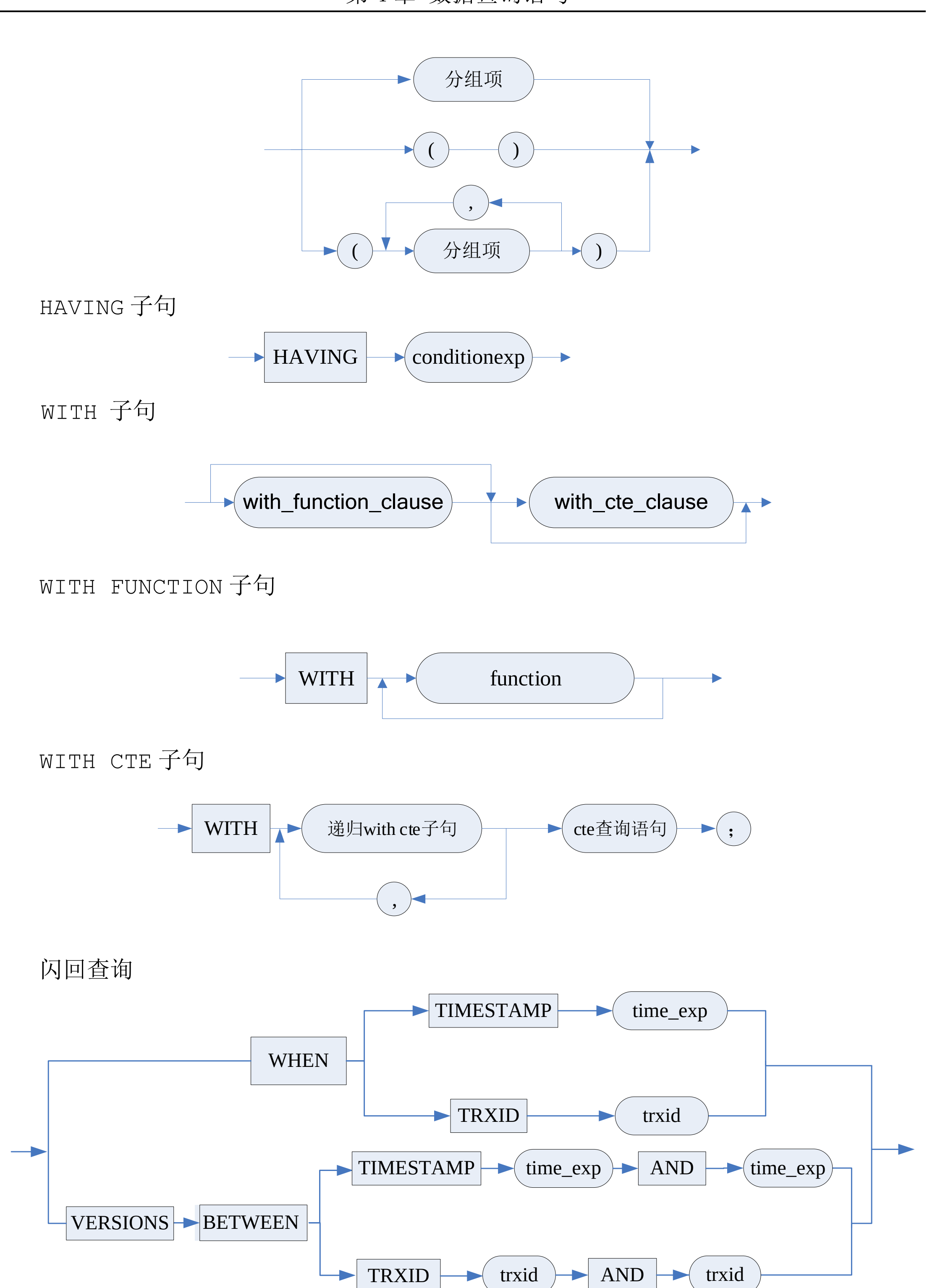

LIMIT子句
ROW LIMIT子句
LIMIT 子句
LIMIT
size
,
size
OFFSET
size
ROW LIMIT 子句
OFFSET
size
ROW
ROWS
FETCH说明
FETCH 说明
FETCH
FIRST
NEXT
size
PERCENT
ROW
ROWS
ONLY
FROM 子句
from
(
,
表引用
)
表引用
普通表1
普通表2
普通表3
ARRAY
数组
连接表
普通表 1


tablename
schemaname
.
viewname
tablename
INDEX
indexname
schemaname
.
PARTITION
(
partitionname
)
SUBPARTITION
(
subpartitionname
)
PIVOT子句
alias
AS
UNPIVOT子句
alias
AS
SAMPLE子句
闪回查询
alias
AS
普通表 2
(
查询表达式
)
PIVOT
子句
alias
AS
UNPIVOT
子句
alias
AS
闪回查询
tablealias
AS
newcolumn
普通表 3
tablename
(
columnname
)
schemaname
.
,
viewname
PIVOT
子句
alias
AS
UNPIVOT
子句
alias
AS
闪回查询
tablealias
AS
(
columnname
)
,
连接表

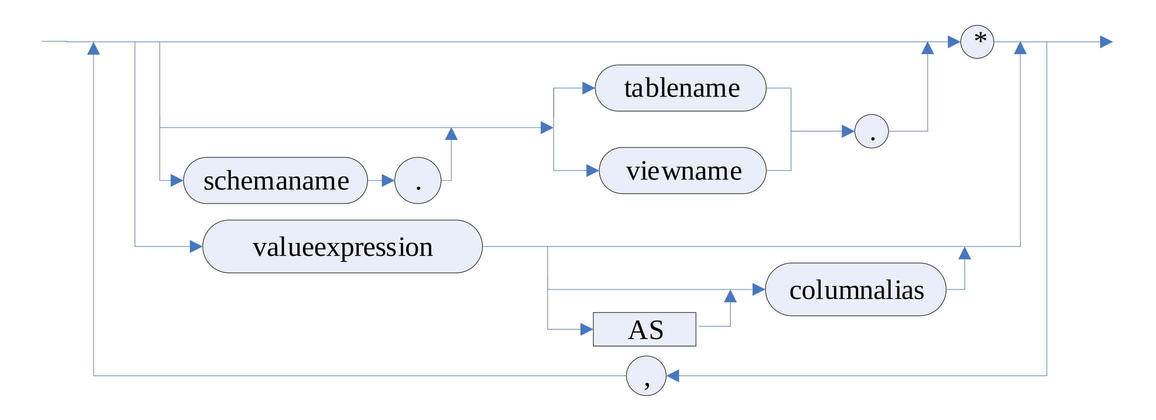

交叉连接
限定连接
(
连接表
)
交叉连接
表引用
CROSS
JOIN
普通表
(
连接表
)
限定连接
表引用
PARTITION BY子句
NATURAL
连接类型
JOIN
普通表
(
连接表
)
PARTITION BY子句
连接类型
INNER
内外连接类型
OUTER
内外连接类型
LEFT
RIGHT
FULL
连接条件
ON
conditionexp
USING
(
columnname
)
,

WHERE 子句

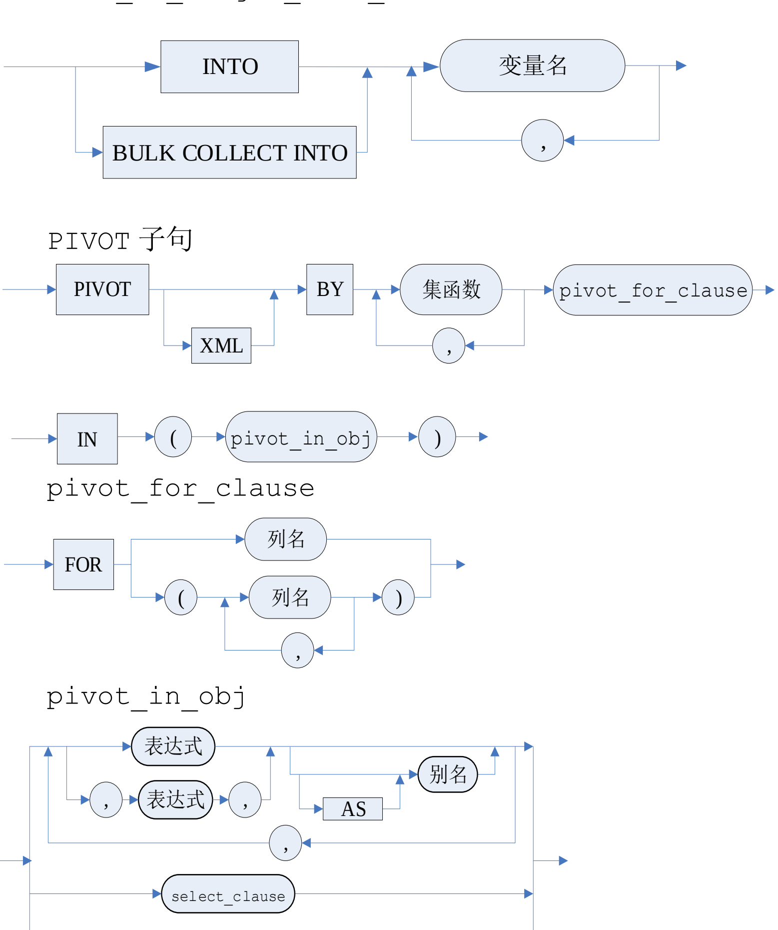

WHERE
conditionexp
WHERE CURRENT OF子句
层次查询子句
CONNECT
BY
NOCYCLE
condition
START
WITH
condition
START
WITH
condition
CONNECT
BY
NOCYCLE
condition
GROUP BY 子句
GROUP
BY
,
分组项
rollup项
cube项
grouping sets项
HAVING
condition
ROLLUP 项
ROLLUP
(
,
分组项
)
CUBE 项
CUBE
(
,
分组项
)
GROUPING SETS 项
GROUPING
SETS
(
,
GROUP项
)
GROUP 项


分组项
(
)
,
(
分组项
)
HAVING 子句
HAVING
conditionexp
WITH 子句
with_function_clause
with_cte_clause
WITH FUNCTION 子句
WITH
function
WITH CTE 子句
WITH
递归with cte子句
cte查询语句
;
,
闪回查询
WHEN
TIMESTAMP
time_exp
TRXID
trxid
VERSIONS
BETWEEN
TIMESTAMP
time_exp
AND
time_exp
TRXID
trxid
AND
trxid

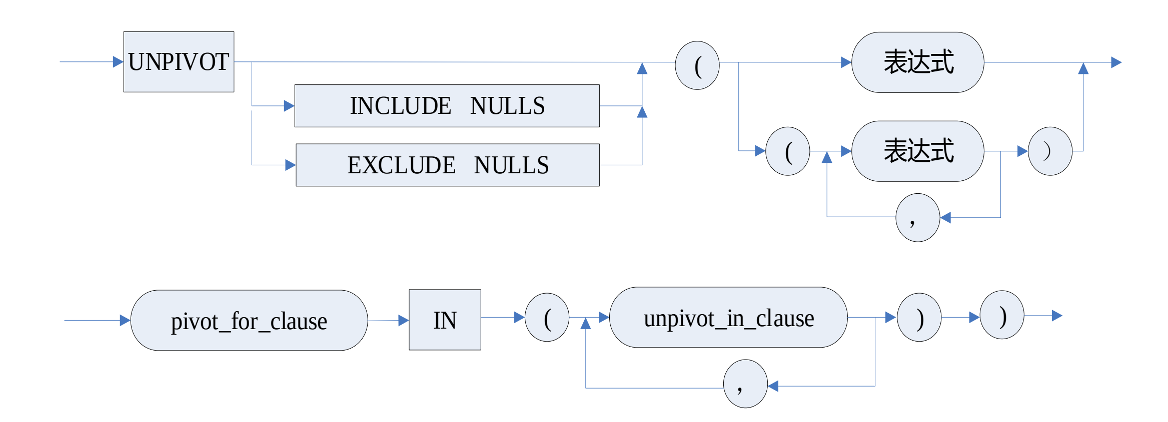
选择列表



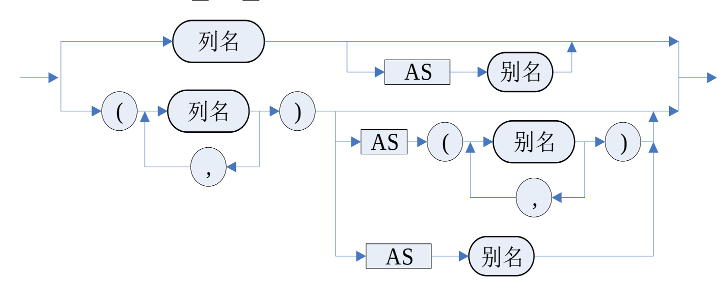
bulk_or_single_into_null



UNPIVOT 子句



unpivot_in_clause



使用说明

1. <选择列表>中最多可包含 1024 个查询项，且查询记录的长度限制不能超过块长的一半；

2. <FROM 子句>中最多可引用 100 张表；

3. WHERE<搜索条件> 用于设置对于行的检索条件。不在规定范围内的任何行都从结果集中去除；

4. 查询语句调用的函数中，不能包含任何增删改操作（包括函数间接调用其它过程\函数产生的增删改操作）；

5. EXCEPT/MINUS/INTERSECT 集合运算中，查询列不能含有 BLOB、CLOB 或 IMAGE、TEXT 等大字段类型；

6. 对于<PIVOT 子句>的使用，存在以下限制：

   1) <pivot_in_clause>中，只有多个表达式才使用()，单个表达式时不支持()；
   2) <pivot_for_clause>中，只有多个列才使用()，单个列时不支持()；
   3) 多个集函数的时候需要为每个集函数定义别名；
   4) <pivot_in_clause>中的表达式仅支持常量表达式；
   5) <pivot_for_clause>中的列个数与<pivot_in_clause>中的表达式个数应一致；
   6) <pivot_in_clause>中的子查询与 ANY 只有在指定 XML 时才生效；
   7) XML 功能的结果集返回类型是 CLOB；
   8) 不支持 COVAR_SAMP、COVAR_POP、LISTAGG/LISTAGG2 集函数；
   9) GROUP BY 项（包括隐含的）不能超过 255 个；
   10) <pivot_in_clause>中表达式的个数与<pivot_for_clause>中集函数的个数乘积不能超过 1024。

7. 对于<UNPIVOT 子句>的使用，存在以下限制：

   1) unpivot_in 指定的列数据类型要保持一致；

2） unpivot_in 的 AS 项必须是常量表达式；
3） 仅支持对单表、视图、DBLINK 进行 UNPIVOT 转换；
4） unpivot_in_clause 中，只有多个表达式才使用()，单个表达式时不支持()；
5） unpivot_for 项个数与 unpivot_in 中的 AS 项表达式数目保持一致；
6） unpivot_val_col_lst 项个数与 unpivot_in 中待转换的列数目保持一致；
7） INI 参数 UNPIVOT_ORDER_FLAG 控制输出结果的顺序，按照不包含在 UNPIVOT 的列进行排序。值含义：1:排序；0:不排序，缺省为 0；
8） UNPIVOT 中自定义列名不能为保留字；
9） unpivot_in_clause 中指定的转换列数目最多 256 列，同时不能超过 INI 参数 N_PARSE_LEVEL 的值；
10）UNPIVOT 不支持 ROLLUP 查询，不能同时存在 PIVOT 子句；
11）UNPIVOT 的列不能是 ROWID\TRXID 列。

4.1 单表查询

SELECT 语句仅从一个表/视图中检索数据，称单表查询。即<FROM 子句>中<普通表>使用的是[<模式名>.]<基表名|视图名>。

4.1.1 简单查询

例 查询所有图书的名字、作者及当前销售价格，并消去重复。

```
SELECT  DISTINCT  NAME, AUTHOR, NOWPRICE  FROM  PRODUCTION .PRODUCT;
```

其中，DISTINCT 保证重复的行将从结果中去除。若允许有重复的元组，改用 ALL 来替换 DISTINCT，或直接去掉 DISTINCT 即可。

查询结果如表 4.1.1 所示。(注：除带 Order By 的查询外，本书所示查询结果中各元组的顺序与实际输出结果中的元组顺序不一定一致。)

表 4.1.1

NAME	AUTHOR	NOWPRICE
长征	王树增	37.7000
工作中无小事	陈满麒	11.4000
红楼梦	曹雪芹，高鹗	15.2000
老人与海	海明威	6.1000
鲁迅文集(小说、散文、杂文)全两册	鲁迅	20.0000
射雕英雄传(全四册)	金庸	21.7000
数据结构(C 语言版)(附光盘)	严蔚敏，吴伟民	25.5000
水浒传	施耐庵，罗贯中	14.3000
突破英文基础词汇	刘毅	11.1000
噼里啪啦丛书(全 7 册)	(日)佐佐木洋子	42.0000

当用户需要查出所有列的数据，且各列的显示顺序与基表中列的顺序也完全相同时，为了方便用户提高工作效率，SQL 语言允许用户将 SELECT 后的<值表达式>省略为*。

```
SELECT * FROM PERSON.PERSON;
```

等价于：

```
SELECT PERSONID, NAME, SEX, EMAIL, PHONE FROM PERSON.PERSON;
```

其查询结果是模式 PERSON 中基表 PERSON 的一份拷贝，结果从略。

例 1）查询 tt 表中有的，kk 表中没有的数据；2）查询 tt 表和 kk 表都有的数据。

```
CREATE TABLE TT(A INT);
INSERT INTO TT VALUES(5);
INSERT INTO TT VALUES(6);
INSERT INTO TT VALUES(7);

CREATE TABLE KK(A INT);
INSERT INTO KK VALUES(5);
INSERT INTO KK VALUES(5);
INSERT INTO KK VALUES(6);
INSERT INTO KK VALUES(8);
```

1） 使用 MINUS 或 EXCEPT 查询 tt 表中有的，kk 表中没有的数据。

```
SELECT * FROM TT MINUS SELECT * FROM KK;  等价于 SELECT * FROM TT EXCEPT SELECT
* FROM KK;
```

其查询结果是：

```
A
7
```

2）使用 INTERSECT 查询 TT 表中和 KK 表中都有的数据。

```
SELECT * FROM TT INTERSECT SELECT * FROM KK;
```

其查询结果是：

```
A
5
6
```

4.1.2 带条件查询

带条件查询是指在指定表中查询出满足条件的元组。该功能是在查询语句中使用 WHERE 子句实现的。WHERE 子句常用的查询条件由谓词和逻辑运算符组成。谓词指明了一个条件，该条件求解后，结果为一个布尔值：真、假或未知。

逻辑算符有：AND，OR，NOT。

谓词包括比较谓词（=、>、<、>=、<=、<>），BETWEEN 谓词、IN 谓词、LIKE 谓词、NULL 谓词、EXISTS 谓词。

1. 使用比较谓词的查询

当使用比较谓词时，数值数据根据它们代数值的大小进行比较，字符串的比较则按序对同一顺序位置的字符逐一进行比较。若两字符串长度不同，短的一方应在其后增加空格，使两串长度相同后再作比较。

例 给出当前销售价格在 10～20 元之间的所有图书的名字、作者、出版社和当前价格。

```
SELECT NAME, AUTHOR, PUBLISHER, NOWPRICE FROM PRODUCTION.PRODUCT
```

```
WHERE  NOWPRICE>=10  AND  NOWPRICE<=20;
```

查询结果如表 4.1.2 所示。

表 4.1.2

NAME	AUTHOR	PUBLISHER	NOWPRICE
红楼梦	曹雪芹，高鹗	中华书局	15.2000
水浒传	施耐庵，罗贯中	中华书局	14.3000
鲁迅文集(小说、散文、杂文)全两册	鲁迅		20.0000
工作中无小事	陈满麒	机械工业出版社	11.4000
突破英文基础词汇	刘毅	外语教学与研究出版社	11.1000

2. 使用 BETWEEN 谓词的查询

例 给出当前销售价格在 10～20 元之间的所有图书的名字、作者、出版社和当前价格。

```
SELECT  NAME, AUTHOR, PUBLISHER, NOWPRICE  FROM  PRODUCTION.PRODUCT
WHERE  NOWPRICE  BETWEEN  10  AND  20;
```

此例查询与上例完全等价，查询结果如上表所示。在 BETWEEN 谓词前面可以使用 NOT，以表示否定。

3. 使用 IN 谓词的查询

谓词 IN 可用来查询某列值属于指定集合的元组。

例 查询出版社为中华书局或人民文学出版社出版的图书名称与作者信息。

```
SELECT  NAME, AUTHOR  FROM  PRODUCTION.PRODUCT
WHERE  PUBLISHER  IN  ('中华书局', '人民文学出版社');
```

查询结果如表 4.1.3 所示。

表 4.1.3

NAME	AUTHOR
红楼梦	曹雪芹， 高鹗
水浒传	施耐庵， 罗贯中
长征	王树增

在 IN 谓词前面也可用 NOT 表示否定。

4. 使用 LIKE 谓词的查询

LIKE 谓词一般用来进行字符串的匹配。我们先用实例来说明 LIKE 谓词的使用方法。

例 查询第一通讯地址中第四个字开始为“关山”且以 202 结尾的地址。

```
SELECT  ADDRESSID,  ADDRESS1,  CITY,  POSTALCODE  FROM  PERSON.ADDRESS
WHERE  ADDRESS1  LIKE  '___关山%202';
```

查询结果如表 4.1.4 所示。

表 4.1.4

ADDRESSID	ADDRESS1	CITY	POSTALCODE
13	洪山区关山春晓 55-1-202	武汉市洪山区	430073
14	洪山区关山春晓 10-1-202	武汉市洪山区	430073
15	洪山区关山春晓 11-1-202	武汉市洪山区	430073

由上例可看出，LIKE 谓词的一般使用格式为：

```
<列名>  LIKE  <匹配字符串常数>
```

其中，<列名>必须是可以转化为字符类型的数据类型的列。对于一个给定的目标行，如果指定列值与由<匹配字符串常数>给出的内容一致，则谓词结果为真。<匹配字符串常数>中的字符可以是一个完整的字符串，也可以是百分号"%"和下划线"_"，"%"和"_"称通配符。"%"代表任意字符串(也可以是空串)；"_"代表任何一个字符。

因此，上例中的 SELECT 语句将从 ADDRESS 表中检索出第一通讯地址中第四个字开始为"关山"且以 202 结尾的地址情况。从该例我们可以看出 LIKE 谓词是非常有用的。使用 LIKE 谓词可以找到所需要的但又记不清楚的那样一些信息。这种查询称模糊查询或匹配查询。为了加深对 LIKE 谓词的理解，下面我们再举几例：

```
ADDRESS1  LIKE  '%洪山%'
```

如果 ADDRESS1 的值含有字符"洪山"，则该谓词取真值。

```
POSTALCODE  LIKE  '43__7_'
```

如果 POSTALCODE 的值由六个字符组成且前两个字符为 43，第五个字符为 7，则该谓词取真值。

```
CITY  LIKE  '%汉阳_'
```

如果 CITY 的值中倒数第三和第二个字为汉阳，则该谓词取真值。

```
ADDRESS1  NOT  LIKE  '洪山%'
```

如果 ADDRESS1 的值的前两个字不是洪山，则该谓词取真值。

阅读以上的例子，读者可能就在想这样一个问题：如果<匹配字符串常数>中所含"%"和"_"不是作通配符，而只是作一般字符使用应如何表达呢？为解决这一问题，SQL 语句对 LIKE 谓词专门提供了对通配符"%"和"_"的转义说明，这时 LIKE 谓语使用格式为：

```
<列名>  LIKE  '<匹配字符串常数>'  [ESCAPE  <转义字符>]
```

其中，<转义字符>指定了一个字符，当该字符出现在<匹配字符串常数>中时，用以指明紧跟其后的"%"或"_"不是通配符而仅作一般字符使用。

例 查询第一通讯地址以 C1_501 结尾的地址，则 LIKE 谓词应为：

```
SELECT  ADDRESSID,  ADDRESS1,  CITY,  POSTALCODE  FROM  PERSON.ADDRESS
WHERE  ADDRESS1  LIKE  '%C1*_501'  ESCAPE  '*';
```

在此例中，*被定义为转义字符，因而在<匹配字符串常数>中*号后的下划线不再作通配符，而是普通字符。

查询结果如表 4.1.5 所示。

表 4.1.5

ADDRESSID	ADDRESS1	CITY	POSTALCODE
16	洪山区光谷软件园 C1_501	武汉市洪山区	430073

为避免错误，转义字符一般不要选通配符"%"、"_"或在<匹配字符串常数>中已出现的字符。

5. 使用.ROW 进行 LIKE 谓词的查询

LIKE 谓词除支持使用列的计算外，还支持通过 ROW 保留字对表或视图进行 LIKE 计算。该查询依次对表或视图中所有字符类型的列进行 LIKE 计算，只要有一列符合条件，则返回 TRUE。

其语法的一般格式为

```
<表名>.ROW  LIKE  <匹配字符串>  [ ESCAPE  <转义字符>]
```

例 查询评论中哪些与曹雪芹有关

```
SELECT * FROM PRODUCTION. PRODUCT_REVIEW WHERE  PRODUCT_REVIEW.ROW LIKE '%
曹雪芹%';
```

该语句等价于

```
SELECT * FROM PRODUCTION. PRODUCT_REVIEW WHERE NAME LIKE '%曹雪芹%' OR EMAIL
LIKE '%曹雪芹%' OR COMMENTS LIKE '%曹雪芹%';
```

6. 使用 NULL 谓词的查询

空值是未知的值。当列的类型为数值类型时，NULL 并不表示 0；当列的类型为字符串类型时，NULL 也并不表示空串。因为 0 和空串也是确定值。NULL 只能是一种标识，表示它在当前行中的相应列值还未确定或未知，对它的查询也就不能使用比较谓词而须使用 NULL 谓词。

例 查询哪些人员的 EMAIL 地址为 NULL。

```
SELECT  NAME, SEX, PHONE  FROM  PERSON.PERSON
WHERE  EMAIL  IS NULL;
```

在 NULL 谓词前，可加 NOT 表示否定。

7. 组合逻辑

可以用逻辑算符(AND，OR，NOT)与各种谓词相组合生成较复杂的条件查询。

例 查询当前销售价格低于 15 元且折扣低于 7 或出版社为人民文学出版社的图书名称和作者。

```
SELECT  NAME, AUTHOR  FROM  PRODUCTION.PRODUCT
WHERE  NOWPRICE < 15 AND DISCOUNT < 7 OR PUBLISHER='人民文学出版社';
```

查询结果如表 4.1.6 所示。

表 4.1.6

NAME	AUTHOR
老人与海	海明威
长征	王树增
工作中无小事	陈满麒

4.1.3 集函数

为了进一步方便用户的使用，增强查询能力，SQL 语言提供了多种内部集函数。集函数又称库函数，当根据某一限制条件从表中导出一组行集时，使用集函数可对该行集作统计操作。

4.1.3.1 函数分类

集函数可分为 10 类：

1. COUNT(*)；
2. 相异集函数 AVG|MAX|MIN|SUM|COUNT(DISTINCT<列名>)；
3. 完全集函数 AVG|MAX|MIN| COUNT|SUM([ALL]<值表达式>)；
4. 方差集函数 VAR_POP、VAR_SAMP、VARIANCE、STDDEV_POP、STDDEV_SAMP、STDDEV；
5. 协方差函数 COVAR_POP、COVAR_SAMP、CORR；
6. 首行函数 FIRST_VALUE；
7. 求区间范围内最大值集函数 AREA_MAX；

8. FIRST/LAST 集函数 AVG|MAX|MIN| COUNT|SUM([ALL] <值表达式>) KEEP (DENSE_RANK FIRST|LAST ORDER BY 子句)；ORDER BY 子句语法参考第 4.7 节；

9. 字符串集函数 LISTAGG/LISTAGG2；

10. 求中位数函数 MEDIAN。

4.1.3.2 使用说明

在使用集函数时要注意以下几点：

1. 相异集函数与完全集函数的区别是：相异集函数是对表中的列值消去重复后再作集函数运算，而完全集函数是对包含列名的值表达式作集函数运算且不消去重复。缺省情况下，集函数均为完全集函数；

2. 集函数中的自变量可以是集函数，但最多只能嵌套 2 层。嵌套分组函数的时候，需要使用 GROUP BY；

3. **AVG**、**SUM** 的参数必须为数值类型；**MAX**、**MIN** 的结果数据类型与参数类型保持一致；对于 **SUM** 函数，如果参数类型为 BYTE、BIT、SMALLINT 或 INTEGER，那么结果类型为 INTEGER，如果参数类型为 NUMERIC、DECIMAL、FLOAT 和 DOUBLE PRECISION，那么结果类型为 DOUBLE PRECISION；COUNT 结果类型统一为 BIGINT；

对于 **AVG** 函数，其参数类型与结果类型对应关系如表 4.1.7 所示：

表 4.1.7

参数类型	结果类型
tinyint	dec(38,6)
smallint	dec(38,6)
int	dec(38,6)
bigint	dec(38,6)
float	double
double	double
dec(x,y)	number

4. 方差集函数中参数 expr 为<列名>或<值表达式>，具体用法如下：

1） **VAR_POP(expr)** 返回 expr 的总体方差。其计算公式为：

$$\frac{SUM(\mathrm{expr}^2)-\dfrac{(SUM(\mathrm{expr}))^2}{COUNT(\mathrm{expr})}}{COUNT(\mathrm{expr})}$$

2） **VAR_SAMP(expr)** 返回 expr 的样本方差，如果 expr 的行数为 1，则返回 NULL。其计算公式为：

$$\frac{SUM(\mathrm{expr}^2)-\dfrac{(SUM(\mathrm{expr}))^2}{COUNT(\mathrm{expr})}}{COUNT(\mathrm{expr})-1}$$

3） **VARIANCE(expr)** 返回 expr 的方差，如果 expr 的行数为 1，则返回为 0，行数大于 1 时，与 var_samp 函数的计算公式一致；

4） **STDDEV_POP(expr)** 返回 expr 的标准差，返回的结果为总体方差的算术平方根，即 var_pop 函数结果的算术平方根。公式如下：

$$\sqrt{\frac{SUM(expr^2)-\frac{(SUM(expr))^2}{COUNT(expr)}}{COUNT(expr)}}$$

5) **STDDEV_SAMP(expr)** 返回 expr 的标准差，返回的结果为样本方差的算术平方根，即 var_samp 函数结果的算术平方根，所以如果 expr 的行数为 1，stddev_samp 返回 NULL；

6) **STDDEV(expr)** 与 stddev_samp 基本一致，差别在于，如果 expr 的行数为 1，stddev 返回 0，即 variance 函数结果的算术平方根。公式如下：

$$\sqrt{\frac{SUM(expr^2)-\frac{(SUM(expr))^2}{COUNT(expr)}}{COUNT(expr)-1}}$$

5. 协方差集函数中参数 expr1 和 expr2 为<列名>或<值表达式>，具体用法如下：

1) **COVAR_POP(expr1, expr2)** 返回 expr1 和 expr2 的总体协方差。其计算公式为：

$$\frac{SUM(expr1*expr2)-\frac{SUM(NVL2(expr2,expr1,NULL))*SUM(NVL2(expr1,expr2,NULL))}{COUNT(NVL2(expr1,expr2,NULL))}}{COUNT(NVL2(expr1,expr2,NULL))}$$

2) **COVAR_SAMP(expr1,expr2)** 返回 expr1 和 expr2 的样本协方差，如果 expr 的行数为 1，则返回 NULL。其计算公式为：

$$\frac{SUM(expr1*expr2)-\frac{SUM(NVL2(expr2,expr1,NULL))*SUM(NVL2(expr1,expr2,NULL))}{COUNT(NVL2(expr1,expr2,NULL))}}{COUNT(NVL2(expr1,expr2,NULL))-1}$$

3) **CORR(expr1, expr2)** 返回 expr1 和 expr2 的相关系数，如果 expr 的行数为 1，则返回 NULL。其计算公式为：

$$\frac{COVAR_POP(expr1,expr2)}{STDDEV_POP(NVL2(expr2,expr1,NULL))*STDDEV_POP(NVL2(expr1,expr2,NULL))}$$

其中 NVL2(expr1, expr2, expr3)表示如果表达式 expr1 非空，NVL2 返回 expr2；如果表达 expr1 为空，NVL2 返回 expr3。

6. **FIRST_VALUE 集函数**，返回查询项的第一行记录；

7. **AREA_MAX(EXP, LOW, HIGH)** 在区间[LOW, HIGH]的范围内取 exp 的最大值。如果 exp 不在该区间内，则返回 LOW 值。如果 LOW 或 HIGH 为 NULL，则返回 NULL。exp 为<变量>、<常量>、<列名>或<值表达式>。参数 exp 类型为 TINYINT、SMALLINT、INT、BIGINT、DEC、FLOAT、DOUBLE、DATE、TIME、DATETIME、BINARY、VARBINARY、INTERVAL YEAR TO MONTH、INTERVAL DAY TO HOUR、TIME WITH TIME ZONE、DATETIEM WITH TIME ZONE。LOW 和 HIGH 的数据类型和 exp 的类型一致，如果不一致，则转换为 exp 的类型，不能转换则报错。AREA_MAX 集函数返回值定义如下：

表 4.1.8 没有 GROUP BY 的情况

EXP 集合	是否有在[LOW, HIGH]区间内的非空值	结果

空集	-	LOW
非空	否	LOW
非空	是	在[LOW,HIGH]区间的最大值

表 4.1.9 有 GROUP BY 的情况

分组前结果	在[LOW, HIGH]区间内是否非空值	结果
空集	--	整个结果为空集
非空集	是	在[LOW,HIGH]区间的最大值
非空集	否	LOW

8. **FIRST/LAST 集函数** 首先根据 SQL 语句中的 GROUP BY 分组(如果没有指定分组则所有结果集为一组)，然后在组内进行排序。根据 FIRST/LAST 计算第一名（最小值）或者最后一名（最大值）的集函数值，排名按照奥林匹克排名法;

9. **LISTAGG/LISTAGG2(exp1, exp2)集函数** 首先根据 SQL 语句中的 GROUP BY 分组(如果没有指定分组则所有结果集为一组)，然后在组内按照 WITHIN GROUP 中的 ORDER BY 进行排序,最后将表达式 exp1 用表达式 exp2 串接起来。LISTAGG2 跟 LISTAGG 的功能是一样的，区别就是 LISTAGG2 返回的是 CLOB 类型，LISTAGG 返回的是 VARCHAR 类型。

LISTAGG 的用法:

```
<LISTAGG>(<参数>[, <参数>]) WITHIN GROUP(<ORDER BY 项>)
```

LISTAGG2 的用法:

```
<LISTAGG2>(<参数>[, <参数>]) WITHIN GROUP(<ORDER BY 项>)
```

10. **MEDIAN 集函数** 当组内排序后，返回组内的中位数。计算过程中忽略空值 NULL。MPP/LPQ 情况下，需要保证组内数据是全的，否则结果错误。MEDIAN()不支持和 DISTINCT 和<KEEP 子句>一起使用。<参数>：参数类型可以是数值类型（INT/DEC)、时间类型（DATETIME/DATE)、时间间隔类型（INTERVAL YEAR TO MONTH)。<参数>暂不支持带时区的时间类型。

MEDIAN 的用法:

```
MEDIAN(<参数>)
```

4.1.3.3 举例说明

下面按集函数的功能分别举例说明。

1. 求最大值集函数 MAX 和求最小值集函数 MIN

例 查询折扣小于 7 的图书中现价最低的价格。

```
SELECT  MIN(NOWPRICE)  FROM  PRODUCTION.PRODUCT
WHERE  DISCOUNT < 7;
```

查询结果为：6.1000

需要说明的是：SELECT 后使用集函数 MAX 和 MIN 得到的是一个最大值和最小值，因而 SELECT 后不能再有列名出现，如果有只能出现在集函数中。如:

```
SELECT  NAME,MIN(NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

DM 系统会报错，因为 NAME 是一个行集合，而最低价格是唯一值。

至于 MAX 的使用格式与 MIN 是完全一样的，读者可以自己举一反三。

2. 求平均值集函数 AVG 和总和集函数 SUM

例 求折扣小于 7 的图书的平均现价。

```
SELECT  AVG(NOWPRICE)  FROM PRODUCTION.PRODUCT WHERE  DISCOUNT < 7;
```

查询结果为：23.15

例 求折扣大于 8 的图书的总价格。

```
SELECT  SUM(NOWPRICE) FROM  PRODUCTION.PRODUCT WHERE   DISCOUNT >8;
```

查询结果为：25.5

3. 求总个数集函数 COUNT

例 查询已登记供应商的个数。

```
SELECT  COUNT(*)  FROM  PURCHASING.VENDOR;
```

查询结果为：12

由此例可看出，COUNT(*)的结果是 VENDOR 表中的总行数，由于主关键字不允许有相同值，因此，它不需要使用保留字 DISTINCT。

例 查询目前销售的图书的出版商的个数。

```
SELECT  COUNT(DISTINCT  PUBLISHER)  FROM  PRODUCTION.PRODUCT;
```

查询结果为：9

由于一个出版商可出版多种图书，因而此例中一定要用 DISTINCT 才能得到正确结果。

4. 求方差集函数 VARIANCE、标准差函数 STDDEV 和样本标准差函数 STDDEV_SAMP

例 求图书的现价方差。

```
SELECT  VARIANCE(NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

查询结果为：1.3664888888888888888888888888888889E2

例 求图书的现价标准差。

```
SELECT STDDEV(NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

查询结果为：11.689692

例 求图书的现价样本标准差。

```
SELECT STDDEV_SAMP(NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

查询结果为：11.689692

5. 求总体协方差集函数 COVAR_POP、样本协方差函数 COVAR_SAMP 和相关系数 CORR

例 求产品原始价格 ORIGINALPRICE 和当头销售价格 NOWPRICE 的总体协方差。

```
SELECT  COVAR_POP(ORIGINALPRICE,  NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

查询结果为：166.226

例 求产品原始价格 ORIGINALPRICE 和当头销售价格 NOWPRICE 的样本协方差。

```
SELECT  COVAR_SAMP(ORIGINALPRICE,  NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

查询结果为：1.8469555555555555555555555555555556E2

例 求产品原始价格 ORIGINALPRICE 和当头销售价格 NOWPRICE 的相关系数。

```
SELECT  CORR(ORIGINALPRICE,  NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

查询结果为：9.6276530968E-001

6.首行函数 FIRST_VALUE

例 返回查询项的首行记录。

```
SELECT FIRST_VALUE(NAME) FROM PRODUCTION.PRODUCT;
```

查询结果为：红楼梦

7. 求区间范围内的最大值函数 AREA_MAX

例 求图书的现价在 20~30 之间的最大值。

```
SELECT area_max(NOWPRICE,20,30) FROM PRODUCTION.PRODUCT;
```

查询结果为：25.5000

8．求 FIRST/LAST 集函数

例 求每个用户最早定的商品中花费最多和最少的金额。

```
select CUSTOMERID, max(TOTAL) keep (dense_rank first order by ORDERDATE) max_val, min(TOTAL) keep (dense_rank first order by ORDERDATE) min_val from SALES.SALESORDER_HEADER group by CUSTOMERID;
```

查询结果为：

```
CUSTOMERID    MAX_VAL    MIN_VAL
-----------        -------     -------
1             36.9000    36.9000
```

9．求 LISTAGG/LISTAGG2 集函数

例 求出版的所有图书。

```
SELECT LISTAGG(NAME, ', ') WITHIN GROUP (ORDER BY NAME) LISTAGG FROM PRODUCTION.PRODUCT;
或
SELECT LISTAGG2(NAME, ', ') WITHIN GROUP (ORDER BY NAME) LISTAGG FROM PRODUCTION.PRODUCT;
```

查询结果为：长征，工作中无小事，红楼梦，老人与海，鲁迅文集(小说、散文、杂文)全两册，射雕英雄传(全四册)，数据结构(C 语言版)(附光盘)，水浒传，突破英文基础词汇，噼里啪啦丛书(全 7 册)

10．求 MEDIAN 集函数

例 求按照 type 分组之后，各组内 nowprice 的中位数。

```
SELECT MEDIAN(nowprice)FROM PRODUCTION.PRODUCT  group by(type);
```

查询结果为：

17.6

18.45

4.1.4 分析函数

分析函数主要用于计算基于组的某种聚合值。

DM 分析函数为用户分析数据提供了一种更加简单高效的处理方式。如果不使用分析函数，则必须使用连接查询、子查询或者视图，甚至复杂的存储过程实现。引入分析函数后，只需要简单的 SQL 语句，并且执行效率方面也有大幅提高。

与集函数的主要区别是，分析函数对于每组返回多行，而集函数对于每个分组只返回一

行。多行形成的组称为窗口，窗口决定了执行当前行的计算范围，窗口的大小可以由组中定义的行数或者范围值滑动。

4.1.4.1 函数分类

分析函数可分为 11 类：

1. COUNT(*)；
2. 完全分析函数 AVG|MAX|MIN| COUNT|SUM([ALL]<值表达式>)，这 5 个分析函数的参数和作为集函数时的参数一致；
3. 方差函数 VAR_POP、VAR_SAMP、VARIANCE、STDDEV_POP、STDDEV_SAMP、STDDEV；
4. 协方差函数 COVAR_POP、COVAR_SAMP、CORR；
5. 首尾函数 FIRST_VALUE、LAST_VALUE；
6. 相邻函数 LAG 和 LEAD；
7. 分组函数 NTILE；
8. 排序函数 RANK、DENSE_RANK、ROW_NUMBER；
9. 百分比函数 PERCENT_RANK、CUME_DIST、RATIO_TO_REPORT、PERCENTILE_CONT、NTH_VALUE；
10. 字符串函数 LISTAGG；
11. 指定行函数 NTH_VALUE。

4.1.4.2 使用说明

1. 分析函数只能出现在选择项或者 ORDER BY 子句中；
2. 分析函数有 DISTINCT 的时候，不允许 ORDER BY 一起使用；
3. 分析函数参数、PARTITION BY 项和 ORDER BY 项中不允许使用分析函数，即不允许嵌套；
4. 当 PARTITION BY 项包含常量表达式时，表示以整个结果集分区；ORDER BY 项中包含常量表达式时，表示以该常量排序，即保持原来结果集顺序；
5. 当省略 PARTITION BY 项时，将所有行视为一个分组；
6. AVG、COUNT、MAX、MIN、SUM 这 5 类分析函数的参数和返回的结果集的数据类型与对应的集函数保持一致，详细参见集函数部分；
7. FIRST 和 LAST 分析函数的参数不允许为 DISTINCT；
8. FIRST_VALUE 分析函数返回组中数据窗口的第一个值，LAST_VALUE 表示返回组中数据窗口 ORDER BY 项相同的最后一个值；
9. FIRST_VALUE/LAST_VALUE/LAG/LEAD/NTH_VALUE 函数支持 RESPECT|IGNORE NULLS 子句，该子句用来指定计算中是否跳过 NULL 值；
10. NTH_VALUE 函数支持 FROM FIRST/LAST 子句，该子句用来指定计算中是从第一行向后还是最后一行向前。

4.1.4.3 具体用法

分析函数的使用，按以下几种情况。

4.1.4.3.1 一般分析函数

分析函数的分析子句语法如下：

```
<分析函数>::=<函数名>(<参数>) OVER (<分析子句>)
```

```
<分析子句>::= [<PARTITION BY 项>] [<ORDER BY 项> [<窗口子句>]]
<PARTITION BY 项>::= PARTITION BY <<常量表达式>| <列名>>
<ORDER BY 项>::= ORDER BY <<常量表达式>| <列名>>
<窗口子句>::=<ROWS | RANGE >
    {
BETWEEN
{<UNBOUNDED PRECEDING>|<CURRENT ROW>|<value_expr <PRECEDING|FOLLOWING> >}
AND
{<UNBOUNDED FOLLOWING>|<CURRENT ROW>|<value_expr <PRECEDING|FOLLOWING> >}
}
| {<UNBOUNDED PRECEDING>|<CURRENT ROW>| <value_expr PRECEDING>}
<函数名>::=见下表
```

窗口子句：不是所有的分析函数都可以使用窗口。其对应关系如下表所示：

表 4.1.10 分析函数窗口列表

序号	函数名	是否为聚合函数	是否允许使用窗口子句
1	AVG	Y	Y
2	CORR	Y	Y
3	COUNT	Y	Y
4	COVAR_POP	Y	Y
5	COVAR_SAMP	Y	Y
6	CUME_DIST	Y	N
7	DENSE_RANK	Y	N
8	FIRST_VALUE	Y	Y
9	LAG	N	N
10	LAST_VALUE	Y	Y
11	LEAD	N	N
12	LISTAGG	Y	N
13	NTH_VALUE	N	N
14	MAX	Y	Y
15	MIN	Y	Y
16	NTILE	N	N
17	PERCENT_RANK	Y	N
18	PERCENTILE_CONT	N	N
19	PERCENTILE_DISC	N	N
20	RANK	Y	N
21	RATIO_TO_REPORT	N	N
22	ROW_NUMBER	N	N
23	STDDEV	Y	Y
24	STDDEV_POP	Y	Y
25	STDDEV_SAMP	Y	Y
26	SUM	Y	Y
27	VAR_POP	Y	Y

28	VAR_SAMP	Y	Y
29	VARIANCE	Y	Y

<窗口子句>参数如下：

1. ROWS | RANGE：用于计算分析函数窗口中计算的范围。窗口从区中第一行开始依次往下滑动。ROWS 表示窗口中的物理行数；RANGE 表示窗口中的逻辑偏移。

1） ROWS：如果 FOLLOWING 是开始点，则结束点必须也是 FOLLOWING；如果 PRECEDING 是结束点，则开始点必须也是 PRECEDING。

2） RANGE：

a) 逻辑偏移值（value_expr）必须为常量、表达式或者正的 NUMERIC 类型数值；

b) <ORDER BY 子句>中如果使用表达式，那么只能声明一个表达式；

c) 如果 value_expr 为 NUMERIC 类型，则 ORDER BY expr 也必须为 NUMERIC 或者 DATE 类型；

d) 如果 value_expr 为 INTERVAL 类型，则 ORDER BY expr 必须为 DATE 数据类型。

2. BETWEEN ... AND：窗口的范围，如果只定义一个分支，其另一个分支为当前行 CURRENT ROW。

1） UNBOUNDED PRECEDING：表示窗口范围从一个分组的第一行开始。只能用于窗口开始点。

2） UNBOUNDED FOLLOWING：表示窗口范围以一个分组的最后一行结束。只能用于窗口结束点。

3. CURRENT ROW：

1） 出现在窗口起始位置，是指窗口以当前行或当前值（依赖于是 ROWS 还是 RANGE）为开始位置。在这种情况下，窗口的结束点不能是 value_expr PRECEDING。

2） 出现在窗口结束位置，是指窗口以当前行或当前值（依赖于窗口类型是 ROWS 还是 RANGE）为结束位置。在这种情况下，窗口的起始点不能是 value_expr FOLLOWING。

4. value_expr PRECEDING 或 value_expr FOLLOWING：

1） 对于 ROWS 或 RANGE，如果 value_expr FOLLOWING 是起始位置，则结束位置也必须是 value_expr FOLLOWING；如果 value_expr PRECEDING 是结束位置，则起始位置必须是 value_expr PRECEDING。

2） 对于 ROWS，value_expr 是一个物理偏移值，且必须是一个可以计算的正数数值型的表达式或常量。如果窗口函数的起始位置到结束位置没有记录，则分析函数的值返回 NULL。

3） 对于 RANGE，value_expr 是一个逻辑偏移，且必须是一个可以计算的正数数值型的表达式、常量或者是时间间隔类型。在<ORDER BY 子句>中，只能指定一个表达式，即排序列不能多于一个，对于 ROWS，则没有这个限制。

特别说明：

1） ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW，表示该组的第一行到当前行；

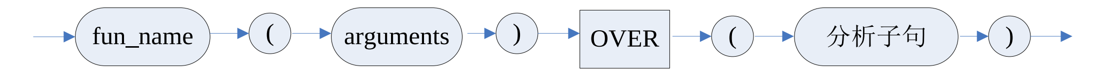
2） ROWS BETWEEN CURRENT ROW AND CURRENT ROW，表示当前行到当前行；

3） ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING，表示该组的第一行到最后一行；

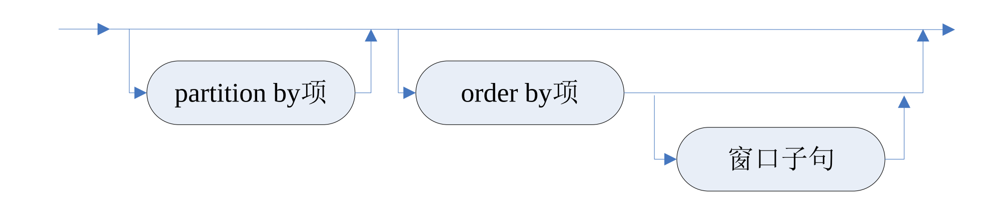
4) ROWS UNBOUNDED PRECEDING，和 1)等价；
5) ROWS CURRENT ROW，和 2)等价；
6) ROWS value_expr PRECEDING，表示 ROWS BETWEEN value_expr PRECEDING AND CURRENT ROW；
7) 如果省略窗口子句，缺省为 ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW。

图例

分析函数语法如下：

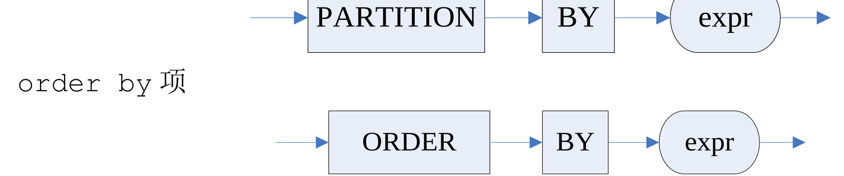


分析子句

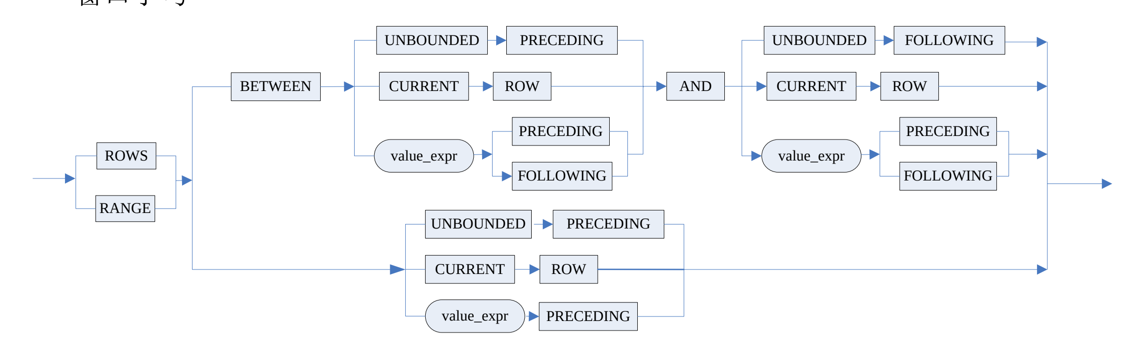


partition by 项

order by 项



窗口子句



4.1.4.3.2 FIRST/LAST 函数

FIRST 和 LAST 作为分析函数时，计算方法和对应的集函数类似，只是一组返回多行。

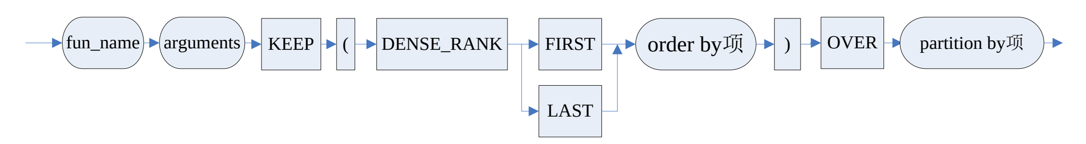
语法格式

```
<函数名><参数> <KEEP 子句> OVER ([<PARTITION BY 项>])
<KEEP 子句>::= KEEP (DENSE_RANK FIRST|LAST <ORDER BY 项>)
<函数名> ::= AVG | MAX | MIN | COUNT | SUM
```

<KEEP 子句>:首先根据<ORDER BY 项>进行排序，然后根据 FIRST/LAST 计算出第一名（最小值）/最后一名（最大值）的函数值，排名按照奥林匹克排名法。

图例

FIRST 和 LAST 分析函数语法如下：



4.1.4.3.3 LAG 和 LEAD 函数

LAG 分析函数表示返回组中和当前行向前相对偏移 offset 行的参数的值，LEAD 方向相反，表示向后相对偏移。如果超出组的总行数范围，则返回 DEFAULT 值。

语法格式

```
<LAG|LEAD> <参数选项 1|参数选项 2> OVER ([<PARTITION BY 项>] <ORDER BY 项>)
<参数选项 1> ::= (<参数>[,<offset>[,<default>]])[<RESPECT|IGNORE> NULLS]
<参数选项 2> ::= (<参数>[<RESPECT|IGNORE> NULLS] [,<offset>[,<default>]])
```

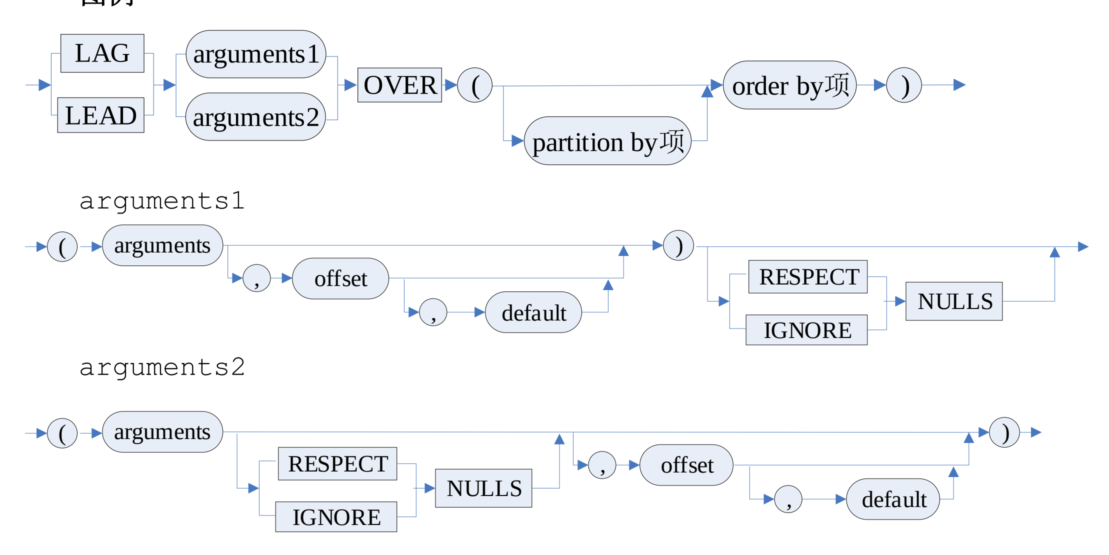
参数

<offset>：为常量或表达式，类型为整型，默认为 1；

<default>：不在 offset 偏移范围内的默认值，为常量或表达式，和 LAG 和 LEAD 的参数数据类型一致；

<RESPECT|IGNORE> NULLS：计算中是否跳过 NULL 值，RESPECT NULLS 为不跳过，IGNORE NULLS 为跳过，默认值为 RESPECT NULLS。

图例



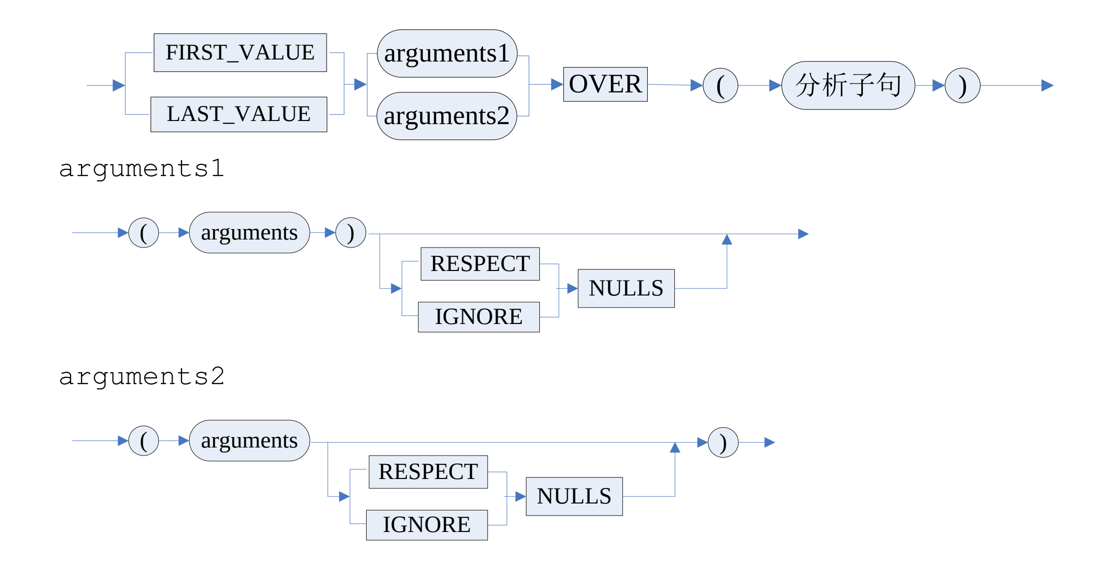
4.1.4.3.4 FIRST_VALUE 和 LAST_VALUE 函数

FIRST_VALUE 返回排序数据集合的第一行，LAST_VALUE 返回其最后一行。

语法格式

```
<FIRST_VALUE|LAST_VALUE> <参数选项 1|参数选项 2> OVER (<分析子句>)
<参数选项 1> ::= (<参数>) [<RESPECT | IGNORE> NULLS ]
<参数选项 2> ::= (<参数> [<RESPECT | IGNORE> NULLS ])
```

图例



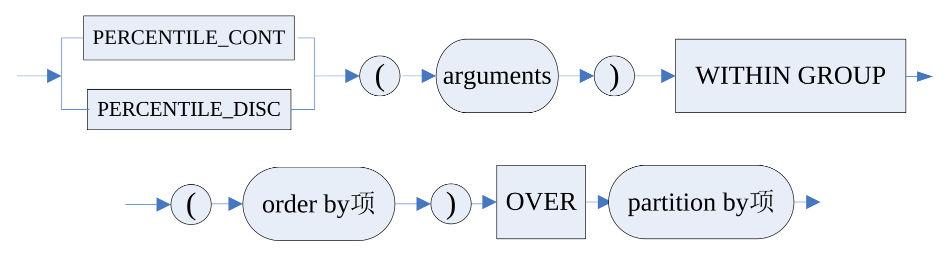
4.1.4.3.5 PERCENTILE_CONT 和 PERCENTILE_DISC 函数

连续百分比 PERCENTILE_CONT 和分布百分比 PERCENTILE_DISC 分析函数。

语法格式

```
<PERCENTILE_CONT|PERCENTILE_DISC> (<参数>) WITHIN GROUP(<ORDER BY 项>) OVER
([<PARTITION BY 项>])
```

图例



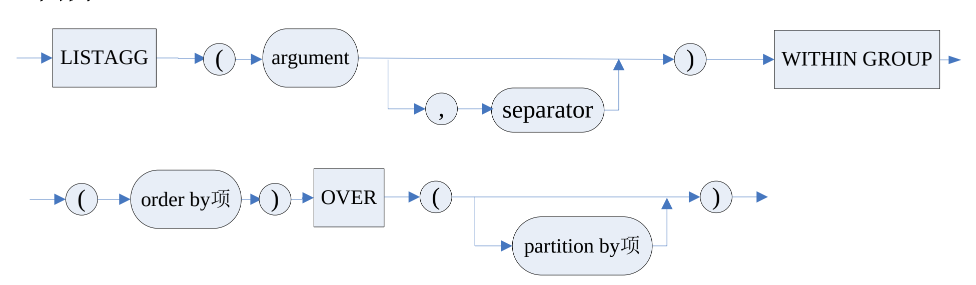
4.1.4.3.6 LISTAGG 函数

字符串分析函数 LISTAGG 按照指定的 PARTITION BY 项进行分组，组内按照 ORDER BY 项排序，将组内的参数通过分隔符拼接起来，返回的结果集行数为组数。

语法格式

```
LISTAGG (参数 [,分隔符]) WITHIN GROUP(<ORDER BY 项>) OVER ([<PARTITION BY 项>])
```

图例



4.1.4.3.7 NTH_VALUE 函数

指定行分析函数 NTH_VALUE 按照指定的 PARTITION BY 项进行分组，组内按照 ORDER BY 项排序，返回组内结果集的指定行的数据。

语法格式

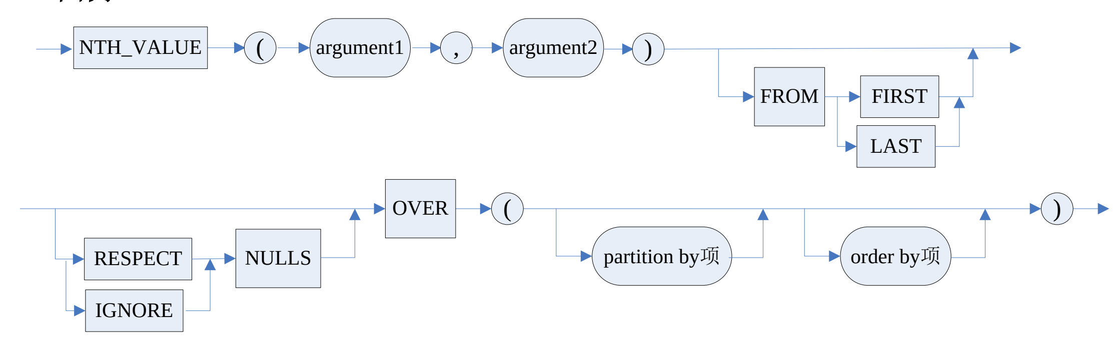
```
NTH_VALUE (参数 1 ,参数 2) [FROM <FIRST | LAST>] [<RESPECT | IGNORE> NULLS] OVER ([<PARTITION BY 项>] [<ORDER BY 项>])
```

参数

1. FROM <FIRST | LAST>：指定组内数据方向，FROM FISRT 指定从第一行往后，FROM LAST 指定从最后一行往前，默认值为 FROM FIRST；

2. <RESPECT|IGNORE> NULLS：计算中是否跳过 NULL 值，RESPECT NULLS 为不跳过，IGNORE NULLS 为跳过，默认值为 RESPECT NULLS。

图例



4.1.4.4 举例说明

下面按分析函数的功能分别举例说明。

1． 最大值 MAX 和最小值 MIN

例 查询折扣大于 7 的图书作者以及最大折扣。

```
SELECT  AUTHOR,  MAX(DISCOUNT) OVER (PARTITION BY AUTHOR)  AS MAX
FROM   PRODUCTION.PRODUCT
WHERE  DISCOUNT > 7;
```

查询结果如表 4.1.11 所示：

表 4.1.11

AUTHOR	MAX
曹雪芹，高鹗	8.0
施耐庵，罗贯中	7.5
严蔚敏，吴伟民	8.5

需要说明的是：如果使用的是集函数 MAX，那么得到的是所有图书中折扣的最大值，并不能查询出作者，使用了分析函数，就可以对作者进行分区，得到每个作者所写的图书中折扣最大的值。MIN 的含义和 MAX 类似。

2． 平均值 AVG 和总和 SUM

例 求折扣小于 7 的图书作者和平均价格。

```
SELECT  AUTHOR,  AVG(NOWPRICE) OVER (PARTITION BY AUTHOR) as AVG
```

```
FROM    PRODUCTION.PRODUCT
WHERE   DISCOUNT < 7;
```

查询结果如表 4.1.12 所示：

表 4.1.12

AUTHOR	AVG
(日)佐佐木洋子	42
陈满麒	11.4
海明威	6.1
金庸	21.7
鲁迅	20
王树增	37.7

例 求折扣大于 8 的图书作者和书的总价格。

```
SELECT  AUTHOR, SUM(NOWPRICE) OVER (PARTITION BY AUTHOR) as SUM
FROM    PRODUCTION.PRODUCT
WHERE   DISCOUNT >8;
```

查询结果如表 4.1.13 所示：

表 4.1.13

AUTHOR	SUM
严蔚敏，吴伟民	25.5

3. 样本个数 COUNT

例 查询信用级别为“很好”的已登记供应商的名称和个数。

```
SELECT  NAME, COUNT(*) OVER (PARTITION BY CREDIT) AS CNT
FROM    PURCHASING.VENDOR
WHERE   CREDIT = 2;
```

查询结果如表 4.1.14 所示：

表 4.1.14

NAME	CNT
长江文艺出版社	2
上海画报出版社	2

由此例可看出，COUNT(*)的结果是 VENDOR 表中的按 CREDIT 分组后的总行数。

4. 分析函数总体协方差 COVAR_POP

例 求产品原始价格 ORIGINALPRICE 和当前销售价格 NOWPRICE 的总体协方差。

```
    SELECT PUBLISHER, COVAR_POP(ORIGINALPRICE, NOWPRICE) OVER(PARTITION BY
PUBLISHER) AS COVAR_POP FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.15 所示：

表 4.1.15

PUBLISHER	COVAR_POP
	0
21 世纪出版社	0

广州出版社	0
机械工业出版社	0
清华大学出版社	0
人民文学出版社	0
上海出版社	0
外语教学与研究出版社	0
中华书局	0
中华书局	0

5． 分析函数样本协方差 COVAR_SAMP

例 求产品原始价格 ORIGINALPRICE 和当前销售价格 NOWPRICE 的样本协方差。

```
SELECT  PUBLISHER, COVAR_SAMP(ORIGINALPRICE, NOWPRICE) OVER(PARTITION BY
PUBLISHER)  AS COVAR_SAMP  FROM  PRODUCTION.PRODUCT;
```

查询结果如表 4.1.16 所示：

表 4.1.16

PUBLISHER	COVAR_SAMP
	NULL
21 世纪出版社	NULL
广州出版社	NULL
机械工业出版社	NULL
清华大学出版社	NULL
人民文学出版社	NULL
上海出版社	NULL
外语教学与研究出版社	NULL
中华书局	0
中华书局	0

6． 相关系数 CORR

例 求产品原始价格 ORIGINALPRICE 和当前销售价格 NOWPRICE 的相关系数。

```
SELECT  PUBLISHER, CORR(ORIGINALPRICE, NOWPRICE) OVER(PARTITION BY PUBLISHER)
AS  CORR  FROM  PRODUCTION.PRODUCT;
```

查询结果如表 4.1.17 所示：

表 4.1.17

PUBLISHER	CORR
	NULL
21 世纪出版社	NULL
广州出版社	NULL
机械工业出版社	NULL
清华大学出版社	NULL

人民文学出版社	NULL
上海出版社	NULL
外语教学与研究出版社	NULL
中华书局	NULL
中华书局	NULL

7. 排名 RANK、DENSE_RANK 和 ROW_NUMBER

例 求按销售额排名的销售代表对应的雇员号和排名。

```
SELECT  EMPLOYEEID,  RANK() OVER (ORDER BY SALESLASTYEAR) AS RANK
FROM  SALES.SALESPERSON;
```

查询结果如表 4.1.18 所示：

表 4.1.18

EMPLOYEEID	RANK
4	1
5	2

RANK()排名函数按照指定 ORDER BY 项进行排名，如果值相同，则排名相同，例如销售额相同的排名相同，该函数使用非密集排名，例如两个第 1 名后，下一个就是第 3 名；与之对应的是 DENSE_RANK()，表示密集排名，例如两个第 1 名之后，下一个就是第 2 名。ROW_NUMBER()表示按照顺序编号，不区分相同值，即从 1 开始编号。

8. FIRST 和 LAST

例 求每个用户最早定的商品中花费最多和最少的金额以及用户当前的花费金额。

```
SELECT CUSTOMERID, TOTAL,
    MAX(TOTAL) KEEP (DENSE_RANK FIRST ORDER BY ORDERDATE) OVER (PARTITION BY
CUSTOMERID) MAX_VAL,
    MIN(TOTAL) KEEP (DENSE_RANK FIRST ORDER BY ORDERDATE) OVER (PARTITION BY
CUSTOMERID) MIN_VAL
FROM SALES.SALESORDER_HEADER;
```

查询结果如表 4.1.19 所示：

表 4.1.19

CUSTOMERID	TOTAL	MAX_VAL	MIN_VAL
1	36.9000	36.9000	36.9000
1	36.9000	36.9000	36.9000

FIRST 和 LAST 分析函数计算方法和对应的集函数类似，作为分析函数时一组返回多行，而集函数只返回一行。

9. FIRST_VALUE 和 LAST_VALUE 分析函数

例 求花费最多和最少金额的用户和花费金额。

```
SELECT NAME, TOTAL,
    FIRST_VALUE(NAME) OVER (ORDER BY TOTAL) FIRST_PERSON,
    LAST_VALUE(NAME) OVER (ORDER BY TOTAL) LAST_PERSON
FROM SALES.SALESORDER_HEADER S,SALES.CUSTOMER C,PERSON.PERSON P
```

```
WHERE S.CUSTOMERID = C.CUSTOMERID AND C.PERSONID = P.PERSONID;
```

查询结果如表 4.1.20 所示：

表 4.1.20

NAME	TOTAL	FIRST_PERSON	LAST_PERSON
刘青	36.9000	刘青	刘青
刘青	36.9000	刘青	刘青

FIRST_VALUE 返回一组中的第一行数据，LAST_VALUE 相反，返回组中的最后一行数据。根据 ORDER BY 项就可以返回需要的列的值。

10. LAG 和 LEAD

例 求当前订单的前一个和下一个订单的销售总额。

```
SELECT
      ORDERDATE,
      LAG(TOTAL, 1, 0) OVER (ORDER BY ORDERDATE) PRV_TOTAL,
      LEAD(TOTAL, 1, 0) OVER (ORDER BY ORDERDATE) NEXT_TOTAL
 FROM  SALES.SALESORDER_HEADER;
```

查询结果如表 4.1.21 所示：

表 4.1.21

ORDERDATE	PRV_TOTAL	NEXT_TOTAL
2007-05-06	0	36.9000
2007-05-07	36.9000	0

LAG 返回当前组的前一个订单日期的 TOTAL 值，如果超出该组，则返回 DEFAULT 值 0。

11. 窗口的使用

例 按照作者分类，求到目前为止图书价格最贵的作者和价格。

```
SELECT AUTHOR,
    MAX(NOWPRICE) OVER(PARTITION  BY  AUTHOR  ORDER  BY  NOWPRICE ROWS
    UNBOUNDED  PRECEDING)  AS  MAX_PRICE
FROM  PRODUCTION.PRODUCT;
```

查询结果如表 4.1.22 所示：

表 4.1.22

AUTHOR	MAX_PRICE
(日)佐佐木洋子	42.0000
曹雪芹,高鹗	15.2000
陈满麒	11.4000
海明威	6.1000
金庸	21.7000
刘毅	11.1000
鲁迅	20.0000
施耐庵，罗贯中	14.3000
王树增	37.7000

严蔚敏，吴伟民	25.5000

分析函数中的窗口限定了计算的范围，ROWS UNBOUNDED PRECEDING 表示该组的第一行开始到当前行，等价于 ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW。

12．标准差 STDDEV

例 求每个出版社图书现价的标准差。

```
SELECT PUBLISHER, STDDEV(NOWPRICE) OVER(PARTITION BY PUBLISHER) AS STDDEV
FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.23 所示：

表 4.1.23

PUBLISHER	STDDEV
	0
21 世纪出版社	0
广州出版社	0
机械工业出版社	0
清华大学出版社	0
人民文学出版社	0
上海出版社	0
外语教学与研究出版社	0
中华书局	0.6363961030678927
中华书局	0.6363961030678927

13．样本标准差 STDDEV_SAMP

例 求每个出版社图书现价的样本标准差。

```
SELECT PUBLISHER, STDDEV_SAMP(NOWPRICE) OVER(PARTITION BY PUBLISHER) AS
STDDEV_SAMP FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.24 所示：

表 4.1.24

PUBLISHER	STDDEV_SAMP
	NULL
21 世纪出版社	NULL
广州出版社	NULL
机械工业出版社	NULL
清华大学出版社	NULL
人民文学出版社	NULL
上海出版社	NULL
外语教学与研究出版社	NULL
中华书局	0.6363961030678927
中华书局	0.6363961030678927

14．总体标准差 STDDEV_POP

例 求每个出版社图书现价的总体标准差。

```
SELECT PUBLISHER, STDDEV_POP (NOWPRICE) OVER(PARTITION BY PUBLISHER) AS
STDDEV_POP FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.25 所示：

表 4.1.25

PUBLISHER	STDDEV_POP
	0
21 世纪出版社	0
广州出版社	0
机械工业出版社	0
清华大学出版社	0
人民文学出版社	0
上海出版社	0
外语教学与研究出版社	0
中华书局	0.45
中华书局	0.45

15. 样本方差 VAR_SAMP

例 求每个出版社图书现价的样本方差。

```
SELECT PUBLISHER, VAR_SAMP(NOWPRICE) OVER(PARTITION BY PUBLISHER) AS
VAR_SAMP FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.26 所示：

表 4.1.26

PUBLISHER	VAR_SAMP
	NULL
21 世纪出版社	NULL
广州出版社	NULL
机械工业出版社	NULL
清华大学出版社	NULL
人民文学出版社	NULL
上海出版社	NULL
外语教学与研究出版社	NULL
中华书局	0.405
中华书局	0.405

16. 总体方差 VAR_POP

例 求每个出版社图书现价的总体方差。

```
SELECT PUBLISHER , VAR_POP(NOWPRICE) OVER(PARTITION BY PUBLISHER) AS
VAR_POP FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.27 所示：

表 4.1.27

PUBLISHER	VAR_POP
	0
21 世纪出版社	0
广州出版社	0
机械工业出版社	0
清华大学出版社	0
人民文学出版社	0
上海出版社	0
外语教学与研究出版社	0
中华书局	0.2025
中华书局	0.2025

17. 方差 VARIANCE

例 求每个出版社图书现价的方差。

```
SELECT PUBLISHER, VARIANCE (NOWPRICE) OVER(PARTITION BY PUBLISHER) AS VARIANCE FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.28 所示：

表 4.1.28

PUBLISHER	VARIANCE
	0
21 世纪出版社	0
广州出版社	0
机械工业出版社	0
清华大学出版社	0
人民文学出版社	0
上海出版社	0
外语教学与研究出版社	0
中华书局	0.405
中华书局	0.405

18. 分组 NTILE

例 根据图书的现价将图书分成三个组。

```
SELECT NAME, NTILE (3) OVER(ORDER BY NOWPRICE) AS NTILE FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.29 所示：

表 4.1.29

NAME	NTILE
老人与海	1
突破英文基础词汇	1
工作中无小事	1

水浒传	1
红楼梦	2
鲁迅文集(小说、散文、杂文)全两册	2
射雕英雄传(全四册)	2
数据结构(C 语言版)(附光盘)	3
长征	3
噼里啪啦丛书(全 7 册)	3

19.排列百分比 PERCENT_RANK

例 求图书的现价排列百分比。

```
SELECT  NAME, PERCENT_RANK()  OVER(ORDER BY NOWPRICE)  AS NTILE FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.30 所示：

表 4.1.30

NAME	NTILE
老人与海	0.0000000000E+000
突破英文基础词汇	1.1111111111E-001
工作中无小事	2.2222222222E-001
水浒传	3.3333333333E-001
红楼梦	4.4444444444E-001
鲁迅文集(小说、散文、杂文)全两册	5.5555555556E-001
射雕英雄传(全四册)	6.6666666667E-001
数据结构(C 语言版)(附光盘)	7.7777777778E-001
长征	8.8888888889E-001
噼里啪啦丛书(全 7 册)	1.0000000000E+000

20.连续百分比对应的值 PERCENTILE_CONT

例 求连续百分比占 0.5 对应的图书现价值。

```
SELECT NAME, PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY NOWPRICE) OVER() AS PERCENTILE_CONT FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.31 所示：

表 4.1.31

NAME	PERCENTILE_CONT
老人与海	17.600000
突破英文基础词汇	17.600000
工作中无小事	17.600000
水浒传	17.600000
红楼梦	17.600000
鲁迅文集(小说、散文、杂文)全两册	17.600000
射雕英雄传(全四册)	17.600000

数据结构(C 语言版)(附光盘)	17.600000
长征	17.600000
噼里啪啦丛书(全 7 册)	17.600000

21. 分布百分比对应的值 PERCENTILE_DISC

例 求分布百分比占 0.5 对应的图书现价值。

```
SELECT NAME, PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY NOWPRICE) OVER()
AS PERCENTILE_DISC FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.32 所示：

表 4.1.32

NAME	PERCENTILE_DISC
老人与海	15.2000
突破英文基础词汇	15.2000
工作中无小事	15.2000
水浒传	15.2000
红楼梦	15.2000
鲁迅文集(小说、散文、杂文)全两册	15.2000
射雕英雄传(全四册)	15.2000
数据结构(C 语言版)(附光盘)	15.2000
长征	15.2000
噼里啪啦丛书(全 7 册)	15.2000

22. 累计百分比 CUME_DIST

例 求图书现价的累计百分比。

```
SELECT NAME, CUME_DIST() OVER(ORDER BY NOWPRICE) AS CUME_DIST FROM
PRODUCTION.PRODUCT;
```

查询结果如表 4.1.33 所示：

表 4.1.33

NAME	CUME_DIST
老人与海	1.0000000000E-001
突破英文基础词汇	2.0000000000E-001
工作中无小事	3.0000000000E-001
水浒传	4.0000000000E-001
红楼梦	5.0000000000E-001
鲁迅文集(小说、散文、杂文)全两册	6.0000000000E-001
射雕英雄传(全四册)	7.0000000000E-001
数据结构(C 语言版)(附光盘)	8.0000000000E-001
长征	9.0000000000E-001
噼里啪啦丛书(全 7 册)	1.0000000000E+000

23. 某一样本值所占百分比 RATIO_TO_REPORT

例 求出版社每种图书现价所占的百分比。

```
SELECT NAME, RATIO_TO_REPORT(NOWPRICE) OVER(PARTITION BY PUBLISHER) AS RATIO_TO_REPORT FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.34 所示:

表 4.1.34

NAME	RATIO_TO_REPORT
老人与海	1.0000000000E+000
突破英文基础词汇	1.0000000000E+000
工作中无小事	1.0000000000E+000
水浒传	1.0000000000E+000
红楼梦	1.0000000000E+000
鲁迅文集(小说、散文、杂文)全两册	1.0000000000E+000
射雕英雄传(全四册)	1.0000000000E+000
数据结构(C 语言版)(附光盘)	1.0000000000E+000
长征	4.8474576271E-001
噼里啪啦丛书(全 7 册)	5.1525423729E-001

24. 组内指定行 NTH_VALUE

例 求每个出版社第二贵的书的价格。

```
SELECT PUBLISHER, NTH_VALUE(NOWPRICE, 2) FROM FIRST RESPECT NULLS OVER(PARTITION BY PUBLISHER ORDER BY NOWPRICE DESC) AS NTH_VALUE FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.35 所示:

表 4.1.35

PUBLISHER	NTH_VALUE
	NULL
21 世纪出版社	NULL
广州出版社	NULL
机械工业出版社	NULL
清华大学出版社	NULL
人民文学出版社	NULL
上海出版社	NULL
外语教学与研究出版社	NULL
中华书局	NULL
中华书局	14.3000

4.1.5 情况表达式

<值表达式>可以为一个<列引用>、<集函数>、<标量子查询>或<情况表达式>等等。

<情况表达式>包括<情况缩写词>和<情况说明>两大类。<情况缩写词>包括函数 NULLIF 和 COALESCE，在 DM 中被划分为空值判断函数。具体函数说明请见 8.4 节。下面详细介绍<情况说明>表达式。

<CASE 情况说明>的语法和语义如下：

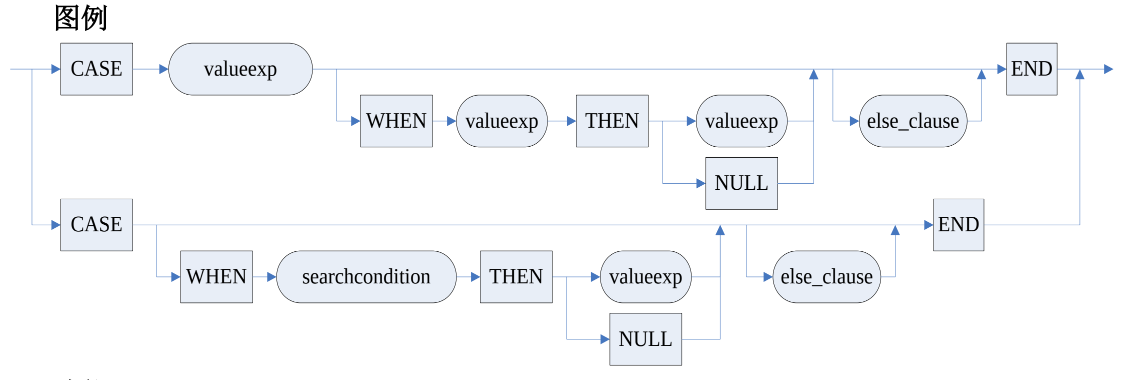
语法格式

```
<情况说明> ::= <简单情况> | <搜索情况>

<简单情况> ::= CASE
<值表达式>
{<简单 WHEN 子句>}
[<ELSE 子句>]
END

<搜索情况> ::= CASE
[<搜索 WHEN 子句>]
[<ELSE 子句>]
END
<简单 WHEN 子句> ::= WHEN <值表达式> THEN <结果>
<搜索 WHEN 子句> ::= WHEN <搜索条件> THEN <结果>
<结果> ::= <值表达式> | NULL
```

图例



功能

指明一个条件值。将搜索条件作为输入并返回一个标量值。

使用说明

1. 在<情况说明>中至少有一个<结果>应该指明<值表达式>；
2. 如果未指明<ELSE 子句>，则隐含 ELSE NULL；
3. <简单情况>中，CASE 运算数的数据类型必须与<简单 WHEN 子句>中的<值表达式>的数据类型是可比较的，且与 ELSE 子句的结果也是可比较的；
4. <情况说明>的数据类型由<结果>中的所有<值表达式>的数据类型确定；
    1) 如果<结果>指明 NULL，则它的值是空值；
    2) 如果<结果>指明<值表达式>，则它的值是该<值表达式>的值。
5. 如果在<情况说明>中某个<搜索 WHEN 子句>的<搜索条件>为真，则<情况说明>的

值是其<搜索条件>为真的第一个<搜索 WHEN 子句>的<结果>的值，并按照<情况说明>的数据类型来转换；

6. <搜索 WHEN 子句>中支持多列，如：

```
    SELECT CASE WHEN (C1,C2) IN (SELECT C1,C2 FROM T2) THEN 1 ELSE
0  END FROM T1;
```

7. 如果在<情况说明>中没有一个<搜索条件>为真，则<情况表达式>的值是其显式或隐式的<ELSE 子句>的<结果>的值，并按照<情况说明>的数据类型来转换；

8. CASE 表达式查询列名为"CASE……END"这部分，最大长度 124 字节，如果大于 124 字节则后面部分截断。

举例说明

例 查询图书信息，如果当前销售价格大于 20 元，返回"昂贵"，如果当前销售价格小于等于 20 元，大于等于 10 元，返回"普通"，如果当前销售价格小于 10 元，返回"便宜"。

```
SELECT NAME,
     CASE
        WHEN NOWPRICE > 20 THEN  '昂贵'
        WHEN NOWPRICE <= 20 AND NOWPRICE >= 10 THEN '普通'
        ELSE '便宜'
     END AS 选择
FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.36 所示：

表 4.1.36

NAME	选择
红楼梦	普通
水浒传	普通
老人与海	便宜
射雕英雄传(全四册)	昂贵
鲁迅文集(小说、散文、杂文)全两册	普通
长征	昂贵
数据结构(C 语言版)(附光盘)	昂贵
工作中无小事	普通
突破英文基础词汇	普通
噼里啪啦丛书(全 7 册)	昂贵

例 在 VERDOR 中如果 NAME 为中华书局或清华大学出版社，且 CREDIT 为 1 则返回"采购"，否则返回"考虑"。

```
    SELECT NAME,
       CASE
            WHEN (NAME = '中华书局' OR NAME = '清华大学出版社') AND CREDIT = 1 THEN
'采购'
            ELSE '考虑'
       END  AS 选择
    FROM PURCHASING.VENDOR;
```

查询结果如表 4.1.37 所示：

表 4.1.37

NAME	选择
上海画报出版社	考虑
长江文艺出版社	考虑
北京十月文艺出版社	考虑
人民邮电出版社	考虑
清华大学出版社	采购
中华书局	采购
广州出版社	考虑
上海出版社	考虑
21 世纪出版社	考虑
外语教学与研究出版社	考虑
社械工业出版社	考虑
文学出版社	考虑

例 在上述表中将 NAME 为中华书局，CREDIT 为 1 的元组返回。

```
SELECT NAME, CREDIT FROM PURCHASING.VENDOR
WHERE NAME IN (SELECT CASE
               WHEN CREDIT = 1 THEN '中华书局'
               ELSE 'NOT EQUAL'
               END
               FROM PURCHASING.VENDOR);
```

查询结果如表 4.1.38 所示：

表 4.1.38

NAME	CREDIT
中华书局	1

例 在上述表中，若 CREDIT 大于 1 则修改该值为 1。

```
UPDATE PURCHASING.VENDOR SET CREDIT = CASE
                        WHEN CREDIT > 1 THEN 1
                        ELSE CREDIT
                        END;
SELECT NAME, CREDIT  FROM PURCHASING.VENDOR;
```

查询结果如表 4.1.39 所示：

表 4.1.39

NAME	CREDIT
上海画报出版社	1
长江文艺出版社	1
北京十月文艺出版社	1
人民邮电出版社	1
清华大学出版社	1
中华书局	1
广州出版社	1

上海出版社	1
21 世纪出版社	1
外语教学与研究出版社	1
机械工业出版社	1
文学出版社	1

4.2 连接查询

如果一个查询包含多个表(>=2)，则称这种方式的查询为连接查询。即<FROM 子句>中使用的是<连接表>。DM 的连接查询方式包括:交叉连接(cross join)、自然连接(natural join)、内连接（inner)、外连接（outer)。下面分别举例说明。

4.2.1 交叉连接

1. 无过滤条件

对连接的两张表记录做笛卡尔集，产生最终结果输出。

例 SALESPERSON 和 EMPLOYEE 通过交叉连接查询 HAIRDATE 和 SALESLASTYEAR。

```
SELECT T1.HAIRDATE, T2.SALESLASTYEAR
FROM  RESOURCES.EMPLOYEE T1 CROSS JOIN SALES.SALESPERSON T2;
```

查询结果如表 4.2.1 所示。

表 4.2.1

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2005-5-2	10.0000
2005-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2004-5-2	10.0000
2004-5-2	20.0000

2. 有过滤条件

对连接的两张表记录做笛卡尔集，根据 WHERE 条件进行过滤，产生最终结果输出。

例 查询性别为男性的员工的姓名与职务。

```
SELECT  T1.NAME, T2.TITLE
FROM PERSON.PERSON T1, RESOURCES.EMPLOYEE T2
WHERE T1.PERSONID = T2.PERSONID AND T1.SEX = 'M';
```

查询结果如表 4.2.2 所示。

表 4.2.2

NAME	TITLE
王刚	销售经理
李勇	采购经理
黄非	采购代表
张平	系统管理员

本例中的查询数据必须来自 PERSON 和 EMPLOYEE 两个表。因此，应在 FROM 子句中给出这两个表的表名(为了简化采用了别名)，在 WHERE 子句中给出连接条件(即要求两个表中 PERSONID 的列值相等)。当参加连接的表中出现相同列名时，为了避免混淆，可在这些列名前加表名前缀。

该例的查询结果是 PERSON 和 EMPLOYEE 在 PERSONID 列上做等值连接产生的。条件“T1.PERSONID = T2.PERSONID”称为连接条件或连接谓词。当连接运算符为“=”号时，称为等值连接，使用其它运算符则称非等值连接。

说明：

1. 连接谓词中的列类型必须是可比较的，但不一定要相同，只要可以隐式转换即可；
2. 不要求连接谓词中的列同名；
3. 连接谓词中的比较操作符可以是>、>=、<、<=、=、< >;
4. WHERE 子句中可同时包含连接条件和其它非连接条件。

4.2.2 自然连接(NATURAL JOIN)

把两张连接表中的同名列作为连接条件，进行等值连接，我们称这样的连接为自然连接。

自然连接具有以下特点：

1. 连接表中存在同名列；
2. 如果有多个同名列，则会产生多个等值连接条件；
3. 如果连接表中的同名列类型不匹配，则报错处理。

例 查询销售人员的入职时间和去年销售总额。

```
SELECT T1.HAIRDATE,  T2.SALESLASTYEAR
FROM  RESOURCES.EMPLOYEE  T1  NATURAL  JOIN  SALES.SALESPERSON  T2;
```

查询结果如表 4.2.3 所示。

表 4.2.3

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000

4.2.3 JOIN … USING

这是自然连接的另一种写法，JOIN 关键字指定连接的两张表，USING 指明连接列。要求 USING 中的列存在于两张连接表中。

例 查询销售人员的入职时间和去年销售总额。

```
SELECT HAIRDATE, SALESLASTYEAR
FROM RESOURCES.EMPLOYEE JOIN SALES.SALESPERSON USING(EMPLOYEEID);
```

查询结果如表 4.2.4 所示。

表 4.2.4

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000

4.2.4 JOIN…ON

这是一种连接查询的常用写法，说明是一个连接查询。JOIN 关键字指定连接的两张表，ON 子句指定连接条件表达式，其中不允许出现 ROWNUM。具体采用何种连接方式，由数据库内部分析确定。

例 查询销售人员的入职时间和去年销售总额。

```
SELECT T1.HAIRDATE,
T2.SALESLASTYEAR
FROM RESOURCES.EMPLOYEE T1 JOIN SALES.SALESPERSON T2
ON T1.EMPLOYEEID=T2.EMPLOYEEID;
```

查询结果如表 4.2.5 所示。

表 4.2.5

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000

4.2.5 自连接

数据表与自身进行连接，我们称这种连接为自连接。

自连接查询至少要对一张表起别名，否则，服务器无法识别要处理的是哪张表。

例 对 PURCHASING.VENDOR 表进行自连接查询

```
SELECT T1.NAME, T2.NAME, T1.ACTIVEFLAG
FROM PURCHASING.VENDOR T1, PURCHASING.VENDOR T2
WHERE T1.NAME = T2.NAME;
```

查询结果如表 4.2.6 所示。

表 4.2.6

NAME	NAME	ACTIVEFLAG
上海画报出版社	上海画报出版社	1

长江文艺出版社	长江文艺出版社	1
北京十月文艺出版社	北京十月文艺出版社	1
人民邮电出版社	人民邮电出版社	1
清华大学出版社	清华大学出版社	1
中华书局	中华书局	1
广州出版社	广州出版社	1
上海出版社	上海出版社	1
21 世纪出版社	21 世纪出版社	1
外语教学与研究出版社	外语教学与研究出版社	1
机械工业出版社	机械工业出版社	1
文学出版社	文学出版社	1

4.2.6 内连接(INNER JOIN)

根据连接条件，结果集仅包含满足全部连接条件的记录，我们称这样的连接为内连接。

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的目录名称和子目录名称。

```
SELECT T1.NAME, T2.NAME
FROM  PRODUCTION.PRODUCT_CATEGORY T1  INNER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.7 所示。

表 4.2.7

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库

计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇
英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读
英语	英语写作
管理	行政管理
管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理
管理	财务管理
少儿	幼儿启蒙
少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志
少儿	少儿英语

因为 PRODUCT_CATEGORY 中的 NAME 为金融的没有对应的子目录，所以结果集中没有金融类的图书信息。

4.2.7 外连接(OUTER JOIN)

外连接对结果集进行了扩展，会返回一张表的所有记录，对于另一张表无法匹配的字段用 NULL 填充返回。DM 数据库支持三种方式的外连接：左外连接、右外连接、全外连接。

外连接中常用到的术语：左表、右表。根据表所在外连接中的位置来确定，位于左侧的表，称为左表；位于右侧的表，称为右表。例如 SELECT * FROM T1 LEFT JOIN T2 ON T1.C1=T2.D1，T1 表为左表，T2 表为右表。

返回所有记录的表根据外连接的方式而定。

1. 左外连接：返回左表所有记录；
2. 右外连接：返回右表所有记录；
3. 全外连接：返回两张表所有记录。处理过程为分别对两张表进行左外连接和右外连接，然后合并结果集。

在左外连接和右外连接中，如果需要对未能匹配的缺失数据进行填充，可以使用分区外连接(PARTITION OUTER JOIN)，分区外连接通常用于处理稀疏数据以得到分析报表。

下面举例说明。

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的所有目录名称和子目录名称，包括没有子目录的目录。

```
SELECT T1.NAME, T2.NAME
FROM  PRODUCTION.PRODUCT_CATEGORY T1  LEFT OUTER JOIN
```

```
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.8 所示。

表 4.2.8

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇
英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读
英语	英语写作
管理	行政管理
管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理
管理	财务管理
少儿	幼儿启蒙
少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志

少儿	少儿英语
金融	NULL

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的目录名称和所有子目录名称，包括没有目录的子目录。

```
SELECT T1.NAME, T2.NAME
FROM  PRODUCTION.PRODUCT_CATEGORY T1  RIGHT OUTER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.9 所示。

表 4.2.9

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇
英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读
英语	英语写作
管理	行政管理
管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理

管理	财务管理
少儿	幼儿启蒙
少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志
少儿	少儿英语
NULL	历史

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的所有目录名称和所有子目录名称。

```
SELECT T1.NAME, T2.NAME
FROM  PRODUCTION.PRODUCT_CATEGORY T1  FULL OUTER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.10 所示。

表 4.2.10

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇
英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读

英语	英语写作
管理	行政管理
管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理
管理	财务管理
少儿	幼儿启蒙
少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志
少儿	少儿英语
金融	NULL
NULL	历史

外连接还有一种写法，在连接条件或 WHERE 条件中，在列后面增加(+)指示左外连接或者右外连接。如果表 A 和表 B 连接，连接条件或者 where 条件中，A 的列带有(+)后缀，则认为是 B LEFT JOIN A。如果用户的(+)指示引起了外连接环，则报错。下面举例说明。

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的目录名称和所有子目录名称，包括没有目录的子目录。

```
SELECT T1.NAME, T2.NAME
FROM PRODUCTION.PRODUCT_CATEGORY T1, PRODUCTION.PRODUCT_SUBCATEGORY T2
WHERE T1.PRODUCT_CATEGORYID(+) = T2.PRODUCT_CATEGORYID;
```

查询结果如上表 4.2.9 所示。

例 新建产品区域销售统计表 SALES.SALESREGION 并插入数据。

```
CREATE TABLE SALES.SALESREGION(REGION CHAR(10), PRODUCTID INT, AMOUNT INT);
INSERT INTO SALES.SALESREGION VALUES('大陆', 2, 19800);
INSERT INTO SALES.SALESREGION VALUES('大陆', 4, 20090);
INSERT INTO SALES.SALESREGION VALUES('港澳台', 6, 5698);
INSERT INTO SALES.SALESREGION VALUES('外国', 9, 3756);
COMMIT;
```

统计每个产品在各个区域的销售量，没有销售则显示 NULL，此时可使用 PARTITON OUTER JOIN 将稀疏数据转为稠密数据。

```
SELECT A.PRODUCTID, B.REGION, B.AMOUNT
FROM PRODUCTION.PRODUCT A LEFT JOIN SALES.SALESREGION B
PARTITION BY(B.REGION) ON A.PRODUCTID=B.PRODUCTID
ORDER BY A.PRODUCTID, B.REGION;
```

查询结果如表 4.2.10 所示。

表 4.2.10

PRODUCTID	REGION	AMOUNT
1	大陆	NULL
1	港澳台	NULL
1	外国	NULL
2	大陆	19800

2	港澳台	NULL
2	外国	NULL
3	大陆	NULL
3	港澳台	NULL
3	外国	NULL
4	大陆	20090
4	港澳台	NULL
4	外国	NULL
5	大陆	NULL
5	港澳台	NULL
5	外国	NULL
6	大陆	NULL
6	港澳台	5698
6	外国	NULL
7	大陆	NULL
7	港澳台	NULL
7	外国	NULL
8	大陆	NULL
8	港澳台	NULL
8	外国	NULL
9	大陆	NULL
9	港澳台	NULL
9	外国	3756
10	大陆	NULL
10	港澳台	NULL
10	外国	NULL

4.3 子查询

在 DM_SQL 语言中，一个 SELECT-FROM-WHERE 语句称为一个查询块，如果在一个查询块中嵌套一个或多个查询块，我们称这种查询为子查询。子查询会返回一个值(标量子查询)或一个表(表子查询)。它通常采用(SELECT…)的形式嵌套在表达式中。子查询语法如下：

```
<子查询> ::= (<查询表达式>)
```

即子查询是嵌入括弧的<查询表达式>，而这个<查询表达式>通常是一个 SELECT 语句。它有下列限制：

1． 在子查询中不得有 ORDER BY 子句；

2． 子查询允许 TEXT 类型与 CHAR 类型值比较。比较时，取出 TEXT 类型字段的最多 8188 字节与 CHAR 类型字段进行比较；如果比较的两字段都是 TEXT 类型，则最多取 300*1024 字节进行比较；

3． 子查询不能包含在集函数中；

4． 在子查询中允许嵌套子查询。

按子查询返回结果的形式，DM 子查询可分为两大类：

1. 标量子查询：只返回一行一列；
2. 表子查询：可返回多行多列。

4.3.1 标量子查询

标量子查询是一个普通 SELECT 查询，它只应该返回一行一列记录。如果返回结果多于一行则会提示单行子查询返回多行，返回结果多于一列则会提示 SELECT 语句列数超长。

下面是一个标量子查询的例子(请先关闭自动提交功能，否则 COMMIT 与 ROLLBACK 会失去效果)：

```
SELECT 'VALUE IS', (SELECT ADDRESS1 FROM PERSON.ADDRESS WHERE ADDRESSID = 1)
FROM PERSON.ADDRESS_TYPE;
-- 子查询只有一列，结果正确

SELECT 'VALUE IS', LEFT((SELECT ADDRESS1 FROM PERSON. ADDRESS WHERE ADDRESSID = 1), 8)  FROM  PERSON.ADDRESS_TYPE;
-- 函数+标量子查询，结果正确

    SELECT 'VALUE IS', (SELECT ADDRESS1, CITY FROM PERSON.ADDRESS WHERE ADDRESSID = 1) FROM  PERSON.ADDRESS_TYPE;
    -- 返回列数不为 1，报错

    SELECT 'VALUES IS', (SELECT ADDRESS1 FROM PERSON.ADDRESS)
    FROM  PERSON.ADDRESS_TYPE;
    -- 子查询返回行值多于一个，报错

     DELETE FROM SALES.SALESORDER_DETAIL;
    SELECT 'VALUE IS', (SELECT ORDERQTY FROM SALES.SALESORDER_DETAIL)
FROM  SALES.CUSTOMER;
    -- 子查询有 0 行，结果返回 NULL

UPDATE PRODUCTION.PRODUCT SET PUBLISHER =
(SELECT NAME FROM PURCHASING.VENDOR WHERE VENDORID = 2)
WHERE PRODUCTID = 5;

UPDATE PRODUCTION.PRODUCT_VENDOR SET STANDARDPRICE =
(SELECT AVG(NOWPRICE) FROM PRODUCTION.PRODUCT)
WHERE PRODUCTID = 1;
-- Update 语句中允许使用标量子查询

INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES
(( SELECT NAME FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE PRODUCT_SUBCATEGORYID= 40));
-- Insert 语句中允许使用标量子查询
```

例如，查询通常价格最小的供应商的名称和最小价格：

```
SELECT NAME, (SELECT  MIN(STANDARDPRICE)
             FROM  PRODUCTION.PRODUCT_VENDOR T1
             WHERE T1.VENDORID = T2.VENDORID)
FROM  PURCHASING.VENDOR T2;
```

4.3.2 表子查询

和标量子查询不同的是，表子查询的查询结果可以是多行多列。

一般情况下，表子查询类似标量子查询，单列构成了表子查询的选择清单，但它的查询结果允许返回多行。可以从上下文中区分出表子查询：在其前面始终有一个只对表子查询的算符：<比较算符>ALL、<比较算符>ANY(或是其同义词<比较算符> SOME)、IN 和 EXISTS。

其中，在 IN/NOT IN 表子查询的情况下，DM 支持查询结果返回多列。

例 查询职务为销售代表的员工的编号、今年销售总额和去年销售总额。

```
SELECT  EMPLOYEEID, SALESTHISYEAR, SALESLASTYEAR
FROM   SALES.SALESPERSON
WHERE  EMPLOYEEID IN
( SELECT  EMPLOYEEID
  FROM  RESOURCES.EMPLOYEE
  WHERE  TITLE = '销售代表'
);
```

查询结果如表 4.3.1 所示。

表 4.3.1

EMPLOYEEID	SALESTHISYEAR	SALESLASTYEAR
4	8.0000	10.0000
5	8.0000	20.0000

该查询语句的求解方式是：首先通过子查询"SELECT EMPLOYEEID FROM RESOURCES.EMPLOYEE WHERE TITLE = '销售代表'"查到职务为销售代表的 EMPLOYEEID 的集合，然后，在 SALESPERSON 表中找到与子查询结果集中的 EMPLOYEEID 所对应员工的 SALESTHISYEAR 和 SALESLASTYEAR。

在带有子查询的查询语句中，通常也将子查询称内层查询或下层查询。由于子查询还可以嵌套子查询，相对于下一层的子查询，上层查询又称为父查询或外层查询。

由于 DM_SQL 语言所支持的嵌套查询功能可以将一系列简单查询构造成复杂的查询，从而有效地增强了 DM_SQL 语句的查询功能。以嵌套的方式构造语句是 DM_SQL 的"结构化"的特点。

需要说明的是：上例的外层查询只能用 IN 谓词而不能用比较算符"="，因为子查询的结果包含多个元组，除非能确定子查询的结果只有一个元组时，才可用等号比较。上例语句也可以用连接查询的方式实现。

```
SELECT T1.EMPLOYEEID, T1.SALESTHISYEAR, T1.SALESLASTYEAR
FROM  SALES.SALESPERSON T1 , RESOURCES.EMPLOYEE T2
WHERE T1.EMPLOYEEID = T2.EMPLOYEEID AND T2.TITLE = '销售代表';
```

例 查询对目录名为小说的图书进行评论的人员名称和评论日期。

采用子查询嵌套方式写出以下查询语句：

```
SELECT  DISTINCT NAME, REVIEWDATE
FROM   PRODUCTION.PRODUCT_REVIEW
WHERE  PRODUCTID  IN
( SELECT  PRODUCTID
  FROM  PRODUCTION.PRODUCT
  WHERE  PRODUCT_SUBCATEGORYID IN
  ( SELECT  PRODUCT_SUBCATEGORYID
    FROM   PRODUCTION.PRODUCT_SUBCATEGORY
    WHERE  PRODUCT_CATEGORYID IN
    ( SELECT  PRODUCT_CATEGORYID
      FROM  PRODUCTION.PRODUCT_CATEGORY
      WHERE  NAME = '小说'
    )
  )
);
```

查询结果如表 4.3.2 所示。

表 4.3.2

NAME	REVIEWDATE
刘青	2007-05-06
桑泽恩	2007-05-06

该语句采用了四层嵌套查询方式，首先通过最内层子查询从 PRODUCT_CATEGORY 中查出目录名为小说的目录编号，然后从 PRODUCT_SUBCATEGORY 中查出这些目录编号对应的子目录编号，接着从 PRODUCT 表中查出这些子目录编号对应的图书的编号，最后由最外层查询查出这些图书编号对应的评论人员和评论日期。

此例也可用四个表的连接来完成。

从上例可以看出，当查询涉及到多个基表时，嵌套子查询与连接查询相比，前者由于是逐步求解，层次清晰，易于阅读和理解，具有结构化程序设计的优点。

在许多情况下，外层子查询与内层子查询常常引用同一个表，如下例所示。

例 查询当前价格低于红楼梦的图书的名称、作者和当前价格。

```
SELECT  NAME, AUTHOR,  NOWPRICE
FROM  PRODUCTION.PRODUCT
WHERE  NOWPRICE < ( SELECT NOWPRICE FROM PRODUCTION.PRODUCT
                     WHERE NAME = '红楼梦');
```

查询结果如表 4.3.3 所示。

表 4.3.3

NAME	AUTHOR	NOWPRICE
水浒传	施耐庵，罗贯中	14.3000
老人与海	海明威	6.1000
工作中无小事	陈满麒	11.4000
突破英文基础词汇	刘毅	11.1000

此例的子查询与外层查询尽管使用了同一表名，但作用是不一样的。子查询是在该表中红楼梦的图书价格，而外查询是在 PRODUCT 表 NOWPRICE 列查找小于该值的集合，从而得到这些值所对应的名称和作者。DM_SQL 语言允许为这样的表引用定义别名：

```
SELECT  NAME, AUTHOR,  NOWPRICE
FROM  PRODUCTION.PRODUCT  T1
WHERE  T1.NOWPRICE < ( SELECT  T2.NOWPRICE
FROM  PRODUCTION.PRODUCT  T2
                      WHERE  T2.NAME = '红楼梦');
```

该语句也可以采用连接方式实现:

```
SELECT  T1.NAME, T1.AUTHOR, T1.NOWPRICE
FROM  PRODUCTION.PRODUCT T1 ,  PRODUCTION.PRODUCT T2
WHERE  T2.NAME = '红楼梦' AND T1.NOWPRICE < T2.NOWPRICE;
```

例 查询图书的出版社和产品供应商名称相同的图书编号和名称。

```
SELECT  T1.PRODUCTID, T1.NAME
FROM  PRODUCTION.PRODUCT T1, PRODUCTION.PRODUCT_VENDOR T2
WHERE  T1.PRODUCTID = T2.PRODUCTID  AND T1.PUBLISHER = ANY
( SELECT  NAME  FROM  PURCHASING.VENDOR T3
  WHERE  T2.VENDORID = T3.VENDORID);
```

其结果如表 4.3.4 所示。

表 4.3.4

PRODUCTID	NAME
1	红楼梦
2	水浒传
3	老人与海
4	射雕英雄传(全四册)
7	数据结构(C 语言版)(附光盘)
8	工作中无小事
9	突破英文基础词汇
10	噼里啪啦丛书(全 7 册)

此例有一点需要注意:子查询的 WHERE 子句涉及到 PRODUCT_VENDOR.VENDORID(即 T2.VENDORID),但是其 FROM 子句中却没有提到 PRODUCT_VENDOR。在外部子查询 FROM 子句中命名了 PRODUCT_VENDOR——这就是外部引用。当一个子查询含有一个外部引用时,它就与外部语句相关联,称这种子查询为相关子查询。

例 查询图书的出版社和产品供应商名称不相同的图书编号和名称。

```
SELECT  T1.PRODUCTID, T1.NAME
FROM  PRODUCTION.PRODUCT T1
WHERE  T1.PUBLISHER <> ALL(SELECT  NAME  FROM  PURCHASING.VENDOR );
```

其结果如表 4.3.5 所示。

表 4.3.5

PRODUCTID	NAME
5	鲁迅文集(小说、散文、杂文)全两册
6	长征

4.3.3 派生表子查询

派生表子查询是一种特殊的表子查询。所谓派生表是指 FROM 子句中的查询表达式，可以以别名对其进行引用。在 SELCET 语句的 FROM 子句中可以包含一个或多个派生表。派生表嵌套层次不能超过 60 层。

说明：在派生表中，如果有重复列名，DM 系统将自动修改其列名。

例 查询每个目录的编号、名称和对应的子目录的数量，并按数量递减排列。

```
SELECT  T1.PRODUCT_CATEGORYID, T1.NAME, T2.NUM
FROM  PRODUCTION.PRODUCT_CATEGORY  T1,
( SELECT PRODUCT_CATEGORYID, COUNT(PRODUCT_SUBCATEGORYID)
  FROM  PRODUCTION.PRODUCT_SUBCATEGORY
  GROUP BY PRODUCT_CATEGORYID
)  AS  T2(PRODUCT_CATEGORYID,NUM)
WHERE T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID
ORDER BY T2.NUM
DESC;
```

查询结果如表 4.3.6 所示。

表 4.3.6

PRODUCT_CATEGORYID	NAME	NUM
3	计算机	8
2	文学	7
6	少儿	6
1	小说	6
5	管理	6
4	英语	6

4.3.4 定量比较

量化符 ALL、SOME、ANY 可以用于将一个<数据类型>的值和一个由表子查询返回的值的集合进行比较。

1. ALL

ALL 定量比较要求的语法如下：

```
<标量表达式> <比较算符> ALL <表子查询>
```

其中：

1) <标量表达式>可以是对任意单值计算的表达式；

2) <比较算符>包括＝、＞、＜、＞＝、＜＝或＜＞。

若表子查询返回 0 行或比较算符对表子查询返回的每一行都为 TRUE，则返回 TRUE。若比较算符对于表子查询返回的至少一行是 FALSE，则 ALL 返回 FALSE。

例 查询没有分配部门的员工的编号、姓名和身份证号码。

```
SELECT  T1.EMPLOYEEID, T2.NAME, T1.NATIONALNO
FROM  RESOURCES.EMPLOYEE T1 ,  PERSON.PERSON T2
```

```
WHERE  T1.PERSONID = T2.PERSONID  AND  T1.EMPLOYEEID <> ALL
        ( SELECT  EMPLOYEEID  FROM  RESOURCES.EMPLOYEE_DEPARTMENT);
```

查询结果如表 4.3.7 所示。

表 4.3.7

EMPLOYEEID	NAME	NATIONALNO
7	王菲	420921197708051523

例 查询比中华书局所供应的所有图书都贵的图书的编号、名称和现在销售价格。

```
SELECT  PRODUCTID, NAME, NOWPRICE
FROM  PRODUCTION.PRODUCT
WHERE  NOWPRICE  > ALL
( SELECT  T1.NOWPRICE
  FROM  PRODUCTION.PRODUCT  T1 , PRODUCTION.PRODUCT_VENDOR T2
  WHERE T1.PRODUCTID = T2.PRODUCTID AND T2.VENDORID =
  ( SELECT  VENDORID  FROM  PURCHASING.VENDOR
    WHERE  NAME = '中华书局'
   )
)
AND  PRODUCTID  <>  ALL
( SELECT  T1.PRODUCTID
  FROM  PRODUCTION.PRODUCT_VENDOR  T1 , PURCHASING.VENDOR  T2
  WHERE  T1.VENDORID =  T2.VENDORID  AND T2.NAME = '中华书局'
);
```

查询结果如表 4.3.8 所示。

表 4.3.8

PRODUCTID	NAME	NOWPRICE
4	射雕英雄传(全四册)	21.7000
5	鲁迅文集(小说、散文、杂文)全两册	20.0000
6	长征	37.7000
7	数据结构(C 语言版)(附光盘)	25.5000
10	噼里啪啦丛书(全 7 册)	42.0000

2. ANY 或 SOME

ANY 或 SOME 定量比较要求的语法如下：

```
<标量表达式> <比较算符> ANY | SOME <表子查询>
```

SOME 和 ANY 是同义词。如果它们对于表子查询返回的至少一行为 TRUE，则返回为 TRUE。若表子查询返回 0 行或比较算符对表子查询返回的每一行都为 FALSE，则返回 FALSE。

ANY 和 ALL 与集函数的对应关系如表 4.3.9 所示。

表 4.3.9　ANY 和 ALL 与集函数的对应关系

	＝	＜＞	＜	＜＝	＞	＞＝
ANY	IN	不存在	＜MAX	＜＝MAX	＞MIN	＞＝MIN
ALL	不存在	NOT IN	＜MIN	＜＝MIN	＞MAX	＞＝MAX

在具体使用时，读者完全可根据自己的习惯和需要选用。

4.3.5 带 EXISTS 谓词的子查询

带 EXISTS 谓词的子查询语法如下：

```
<EXISTS 谓词> ::= [NOT] EXISTS <表子查询>
```

EXISTS 判断是对非空集合的测试并返回 TRUE 或 FALSE。若表子查询返回至少一行，则 EXISTS 返回 TRUE，否则返回 FALSE。若表子查询返回 0 行，则 NOT EXISTS 返回 TRUE，否则返回 FALSE。

例 查询职务为销售代表的员工的编号和入职时间。

```
SELECT  T1.EMPLOYEEID , T1.STARTDATE
FROM  RESOURCES.EMPLOYEE_DEPARTMENT T1
WHERE  EXISTS
( SELECT  *  FROM  RESOURCES.EMPLOYEE T2
  WHERE  T2.EMPLOYEEID = T1.EMPLOYEEID  AND T2.TITLE = '销售代表');
```

查询结果如表 4.3.10 所示。

表 4.3.10

EMPLOYEEID	STARTDATE
5	2005-02-01
4	2005-02-01

此例查询需要 EMPLOYEE_DEPARTMENT 表和 EMPLOYEE 表中的数据，其执行方式为：首先在 EMPLOYEE_DEPARTMENT 表的第一行取 EMPLOYEEID 的值为 2，这样对内层子查询则为：

```
(SELECT  *  FROM  RESOURCES.EMPLOYEE  T2
WHERE  T2.EMPLOYEEID='2'  AND  T2.TITLE='销售代表');
```

在 EMPLOYEE 表中，不存在满足该条件的行，子查询返回值为假，说明不能取 EMPLOYEE_DEPARTMENT 表的第一行作为结果。系统接着取 EMPLOYEE_DEPARTMENT 表的第二行，又得到 EMPLOYEEID 的值为 4，执行内层查询，此时子查询返回值为真，说明可以取该行作为结果。重复以上步骤……。只有外层子查询 WHERE 子句结果为真时，方可将 EMPLOYEE_DEPARTMENT 表中的对应行送入结果表，如此继续，直到把 EMPLOYEE_DEPARTMENT 表的各行处理完。

从以上分析得出，EXISTS 子查询的查询结果与外表相关，即连接条件中包含内表和外表列，我们称这种类型的子查询为相关子查询；反之，子查询的连接条件不包含外表列，即查询结果不受外表影响，我们称这种类型的子查询为非相关子查询。

4.3.6 多列表子查询

为了满足应用需求，DM 数据库扩展了子查询功能，目前支持多列 IN / NOT IN 子查询。

子查询可以是值列表或者查询块。

例 查询活动标志为 1 且信誉为 2 的供应商编号和名称。

```
SELECT  VENDORID, NAME
FROM  PURCHASING.VENDOR
WHERE  (ACTIVEFLAG, CREDIT) IN ((1, 2));
```

查询结果如表 4.3.11 所示。

表 4.3.11

VENDORID	NAME
1	上海画报出版社
2	长江文艺出版社

上例中子查询的选择清单为多列，而看到子查询算符后面跟着的形如((1，2))的表达式我们称之为多列表达式链表，这个多列表达式链表以一个或多个多列数据集构成的集合构成。上述的例子中的多列表达式链表中的元素有两个。

例 查询作者为海明威且出版社为上海出版社或作者为王树增且出版社为人民文学出版社的图书名称和现在销售价格。

```
SELECT  NAME, NOWPRICE
FROM  PRODUCTION.PRODUCT
WHERE  (AUTHOR, PUBLISHER)  IN
(( '海明威', '上海出版社'), ('王树增', '人民文学出版社'));
```

查询结果如表 4.3.12 所示。

表 4.3.12

NAME	NOWPRICE
老人与海	6.1000
长征	37.7000

子查询为值列表时，需要注意以下三点：

1. 值列表需要用括号；
2. 值列表之间以逗号分割；
3. 值列表的个数与查询列个数相同。

子查询为查询块的情况如下例所示：

例 查询由采购代表下的供应商是清华大学出版社的订单的创建日期、状态和应付款总额。

```
SELECT  ORDERDATE, STATUS, TOTAL
FROM   PURCHASING.PURCHASEORDER_HEADER
WHERE  (EMPLOYEEID, VENDORID)  IN
(SELECT  T1.EMPLOYEEID,  T2.VENDORID
         FROM  RESOURCES.EMPLOYEE  T1,  PURCHASING.VENDOR  T2
         WHERE  T1.TITLE = '采购代表' AND T2.NAME = '清华大学出版社');
```

查询结果如表 4.3.13 所示。

表 4.3.13

ORDERDATE	STATUS	TOTAL
2006-07-21	1	6400

由例子可以看到，WHERE 子句中有两个条件列，NOT IN 子查询的查询项也由两列构成。

DM 对多列子查询的支持，满足了更多的应用场景。

4.4 WITH 子句

WITH 子句语法如下：

```
<WITH 子句> ::= [<WITH FUNCTION 子句>] [<WITH CTE 子句>]
```

4.4.1 WITH FUNCTION 子句

WITH FUNCTION 子句用于在 SQL 语句中临时声明并定义存储函数，这些存储函数可以在其作用域内被引用。相比模式对象中的存储函数，通过 WITH FUNCTION 定义的存储函数在对象名解析时拥有更高的优先级。

和公用表表达式 CTE 类似，WITH FUNCTION 定义的存储函数对象也不会存储到系统表中，且只在当前 SQL 语句内有效。

WITH FUNCTION 子句适用于偶尔需要使用存储过程的场景。和模式对象中的存储函数相比，它可以清楚地看到函数定义并避免了 DDL 操作带来的开销。

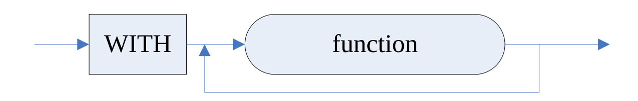
语法格式

```
WITH <函数> {<函数>}
```

参数

1. <函数> 语法遵照《DM8_SQL 程序设计》中存储过程的语法规则。

图例



语句功能

供用户定义同一语句内临时使用的存储函数。

使用说明

1. <WITH FUNCTION 子句>中定义的函数的作用域为<WITH 子句>所在的查询表达式内；
2. 同一<WITH FUNCTION 子句>中函数名不得重复；
3. <WITH FUNCTION 子句>中定义的函数不能是外部函数；
4. 该语句的使用者并不需要 CREATE PROCEDURE 数据库权限。

举例说明

例 WITH FUNCTION 中定义的函数优先级高于模式对象的例子。

```
WITH FUNCTION f1(C INT) RETURN INT AS BEGIN RETURN C * 10; END;
SELECT f1(5236) FROM DUAL;
/
```

查询结果：

```
52360
```

例 WITH FUNCTION 和公用表表达式混合的例子。

```
WITH FUNCTION f21(C1 INT) RETURN INT AS BEGIN RETURN C1; END;
SELECT f21(1) FROM dual WHERE 100 IN
(
  WITH FUNCTION f22(C1 INT) RETURN INT AS BEGIN RETURN C1 + 2; END;
```

```
        FUNCTION f23(C1 INT) RETURN INT AS BEGIN RETURN C1 - 2; END;
                v21(C) AS (SELECT 50 FROM dual)
  SELECT f22(C) +f23(C) FROM v21
);
/
```

查询结果：

```
1
```

4.4.2 WITH CTE 子句

嵌套 SQL 语句如果层次过多，会使 SQL 语句难以阅读和维护。如果将子查询放在临时表中，会使 SQL 语句更容易维护，但同时也增加了额外的 I/O 开销，因此，临时表并不太适合数据量大且频繁查询的情况。为此，在 DM 中引入了公用表表达式(CTE，COMMON TABLE EXPRESSION)，使用 CTE 可以提高 SQL 语句的可维护性，同时 CTE 要比临时表的效率高很多。

CTE 与派生表类似，具体表现在不存储为对象，并且只在查询期间有效。与派生表的不同之处在于，CTE 可自引用，还可在同一查询中引用多次。

WITH CTE 子句会定义一个公用表达式，该公用表达式会被整个 SQL 语句所用到。它可以有效提高 SQL 语句的可读性，也可以用在 UNION ALL 中，作为提供数据的部分。

WITH CTE 子句根据 CTE 是否递归执行 CTE 自身，DM 将 WITH CTE 子句分为递归 WITH 和非递归 WITH 两种情况。

4.4.2.1 公用表表达式的作用

公用表表达式（CTE）是一个在查询中定义的临时命名结果集，将在 FROM 子句中使用它。每个 CTE 仅被定义一次（但在其作用域内可以被引用任意次），并且在该查询生存期间将一直生存，而且可以使用 CTE 来执行递归操作。

因为 UNION ALL 的每个部分可能相同，但是如果每个部分都去执行一遍的话，则成本太高，所以可以使用 WITH CTE 子句，则只要执行一遍即可。如果 WITH CTE 子句所定义的表名被调用两次以上，则优化器会自动将 WITH CTE 子句所获取的数据放入一个临时表里，如果只是被调用一次则不会，很多查询通过这种方法都可以提高速度。

4.4.2.2 非递归 WITH 的使用

语法格式

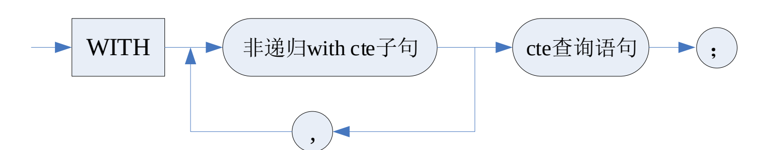
```
WITH <非递归 with cte 子句>{,<非递归 with cte 子句>}<cte 查询语句>;
<非递归 with cte 子句>:: = <公用表表达式的名称> [<列名> ({,<列名>})]  AS  ( <公用表表达式子查询语句>)
```

参数

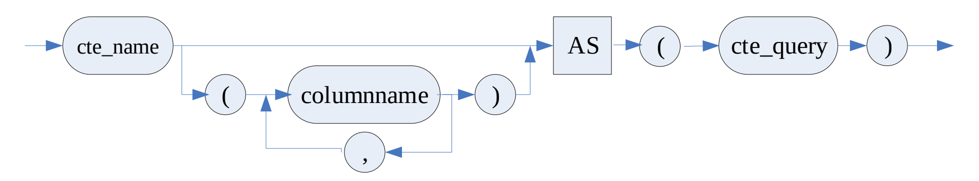
1. <列名>　指明被创建的公用表表达式中列的名称；

2. <公用表表达式子查询语句> 标识公用表表达式所基于的表的行和列，其语法遵照 SELECT 语句的语法规则。

图例



<非递归 with cte 子句>:



语句功能

供用户定义非递归公用表表达式，也就是非递归 WITH 语句。

使用说明

1. < 公用表表达式的名称 >必须与在同一 WITH 子句中定义的任何其他公用表表达式的名称不同，但公用表表达式名可以与基表或基视图的名称相同。在查询中对公用表表达式名的任何引用都会使用公用表表达式，而不使用基对象；

2. 在一个 CTE 定义中不允许出现重复的列名。指定的列名数必须与<公用表表达式子查询语句>结果集中列数匹配。只有在查询定义中为所有结果列都提供了不同的名称时，列名称列表才是可选的；

3. <公用表表达式子查询语句>指定一个结果集填充公用表表达式的 SELECT 语句。除了 CTE 不能定义另一个 CTE 以外，<公用表表达式子查询语句> 的 SELECT 语句必须满足与创建视图时相同的要求；

4. <cte 查询语句> SELECT 查询语句。此处，语法上<cte 查询语句>支持任意 SELECT 语句，但是对于 CTE 而言，只有<cte 查询语句>中使用<公用表表达式的名称>，CTE 才有意义。

权限

该语句的使用者必须对< cte 查询语句>中的每个表均具有 SELECT 权限。

举例说明

公用表表达式可以认为是在单个 SELECT、INSERT、UPDATE、DELETE 或 CREATE VIEW 语句的执行范围内定义的临时结果集。

例 创建一个表 TEST1 和表 TEST2，并利用公用表表达式对它们进行连接运算。

```
CREATE TABLE TEST1(I INT);
INSERT INTO TEST1 VALUES(1);
INSERT INTO TEST1 VALUES(2);

CREATE TABLE TEST2(J INT);
INSERT INTO TEST2 VALUES(5);
INSERT INTO TEST2 VALUES(6);
INSERT INTO TEST2 VALUES(7);

WITH CTE1(K) AS(SELECT I FROM TEST1 WHERE I > 1),
CTE2(G) AS(SELECT J FROM TEST2 WHERE J > 5)
SELECT K, G FROM CTE1, CTE2;
```

运算结果：

```
        K               G
1       2               6
2       2               7
```

例 利用公用表表达式将表 TEST1 中的记录插入到 TEST2 表中。

```
INSERT INTO TEST2 WITH CTE1 AS(SELECT * FROM TEST1)
SELECT * FROM CTE1;

SELECT * FROM TEST2;
```

运行结果：

```
     J
-----------------
     5
     6
     7
     1
     2
```

4.4.2.3 递归 WITH 的使用

递归 WITH 是一个重复执行初始 CTE 以返回数据子集直到获取完整结果集的公用表表达式。递归 WITH 通常用于返回分层数据。

语法格式：

```
WITH <递归with cte子句>{,<递归with cte子句>}<cte查询语句>;
<递归 with cte 子句>::=<公用表表达式的名称> (<列名>{,<列名>}) AS (<定位点成员>
UNION ALL <递归成员>)
```

参数

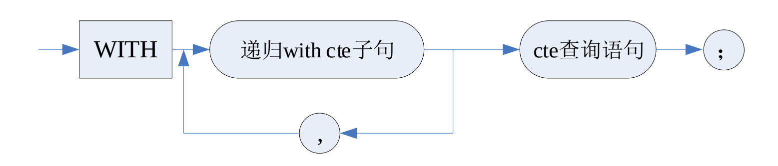
1. <列名> 指明被创建的递归 WITH 中列的名称；各列不能同名，列名和 AS 后的列名没有关系，类似建视图时为视图指定的列别名。

2. <定位点成员> 任何不包含<公用表表达式的名称>的 SELECT 查询语句，可以 UNION ALL、 UNION、 INTERSECT 或 MINUS。定位点成员的查询结果集是递归成员迭代的基础。

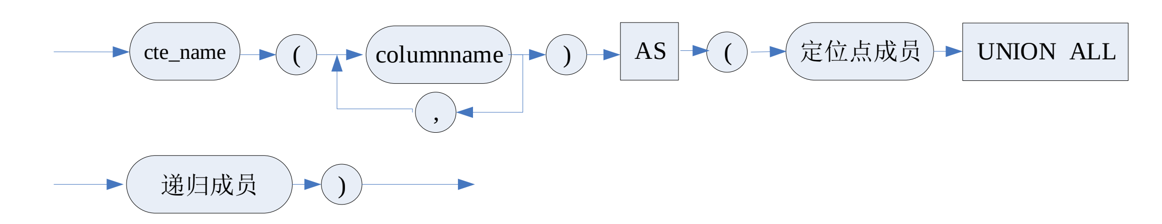
3. <递归成员> 引用<公用表表达式的名称>的 SELECT 查询语句。递归成员通过引用自身<公用表表达式的名称>反复迭代执行，下一次迭代的数据基于上一次迭代的查询结果，当且仅当本次迭代结果为空集时才终止迭代。

4. <cte 查询语句> SELECT 查询语句。此处，语法上<cte 查询语句>支持任意 SELECT 语句，但是对于 CTE 而言，只有<cte 查询语句>中使用<公用表表达式的名称>，CTE 才有意义。

图例



<递归 with cte 子句>：



语句功能

供用户定义递归公用表表达式，也就是递归 WITH 语句。

与递归 WITH 有关的两个 INI 参数为 CTE_MAXRECURSION 和 CTE_OPT_FLAG。CTE_MAXRECURSION 用来指定递归 CTE 迭代层次，取值范围[1，ULINT_MAX]，默认 100。CTE_OPT_FLAG 用来指定递归 WITH 相关子查询是否转换为 WITH FUCNTION 优化，取值 0 或 1，默认 1。

递归 WITH 的执行流程如下：

1. 将递归 WITH 拆分为定位点成员和递归成员；
2. 运行定位点成员，创建第一个基准结果集 (T0)；
3. 运行递归成员，将 TI 作为输入（初始 i=0），将 TI+1 作为输出，I=I++；
4. 重复步骤 3，直到返回空集；
5. 返回结果集为 T0 到 TN 执行 UNION ALL 的结果。

使用说明

1. <公用表表达式的名称>在定位点成员中不能出现。<公用表表达式的名称>在递归成员中有且只能引用一次；
2. 递归成员中不能包含下面元素：
   - DISTINCT；
   - GROUP BY；
   - 集函数，但支持分析函数；
   - <公用表表达式的名称>不能在<递归 with cte 子句>中使用；
   - <公用表表达式的名称>不能作为<递归成员>中外连接 OUTER JOIN 的右表。
3. <递归成员>中列的数据类型必须与定位点成员中相应列的数据类型兼容。

举例说明

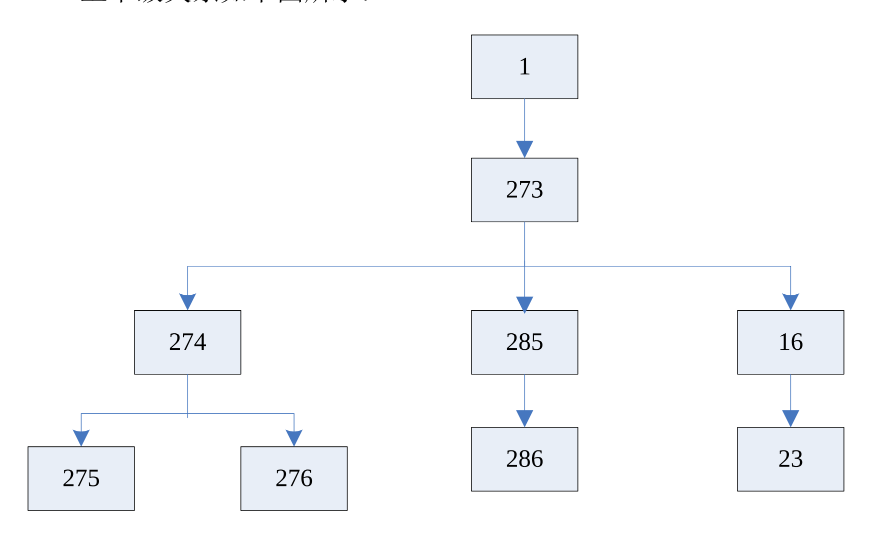
```
DROP TABLE MYEMPLOYEES;
CREATE TABLE MYEMPLOYEES(
EMPLOYEEID  SMALLINT,
    FIRST_NAME  VARCHAR2 (30) NOT NULL,
    LAST_NAME  VARCHAR2 (40) NOT NULL,
    TITLE       VARCHAR2 (50) NOT NULL,
    DEPTID      SMALLINT NOT NULL,
    MANAGERID   INT NULL);

INSERT INTO MYEMPLOYEES VALUES (1, 'KEN', 'SANCHEZ', 'CHIEF EXECUTIVE OFFICER',
16, NULL);
INSERT INTO MYEMPLOYEES VALUES (273, 'BRIAN', 'WELCKER', 'VICE PRESIDENT OF SALES',
3, 1);
INSERT INTO MYEMPLOYEES VALUES (274, 'STEPHEN','JIANG', 'NORTH AMERICAN SALES
MANAGER',3, 273);
```

```
INSERT INTO MYEMPLOYEES VALUES (275, 'MICHAEL', 'BLYTHE', 'SALES REPRESENTATIVE',
3, 274);
INSERT INTO MYEMPLOYEES VALUES (276, 'LINDA', 'MITCHELL', 'SALES REPRESENTATIVE',
3, 274);
INSERT INTO MYEMPLOYEES VALUES (285, 'SYED', 'ABBAS', 'PACIFIC SALES MANAGER',
3, 273);
INSERT INTO MYEMPLOYEES VALUES (286, 'LYNN', 'TSOFLIAS', 'SALES REPRESENTATIVE',
3, 285);
INSERT INTO MYEMPLOYEES VALUES (16, 'DAVID', 'BRADLEY', 'MARKETING MANAGER', 4,
273);
INSERT INTO MYEMPLOYEES VALUES (23, 'MARY', 'GIBSON', 'MARKETING SPECIALIST',
4, 16);

commit;
```

上下级关系如下图所示：




```
WITH DIRECTREPORTS(MANAGERID, EMPLOYEEID, TITLE, DEPTID) AS
(SELECT MANAGERID, EMPLOYEEID, TITLE, DEPTID
   FROM MYEMPLOYEES
   WHERE MANAGERID IS NULL   --定位点成员
   UNION ALL
   SELECT E.MANAGERID, E.EMPLOYEEID, E.TITLE, E.DEPTID
   FROM MYEMPLOYEES E
INNER JOIN DIRECTREPORTS D
ON E.MANAGERID = D.EMPLOYEEID  --递归成员
)
SELECT MANAGERID, EMPLOYEEID, TITLE  FROM DIRECTREPORTS;
```

递归调用执行步骤：

（1）产生定位点成员

```
MANAGERID  EMPLOYEEID  TITLE
---------  ----------  ------------------
NULL    1   CHIEF EXECUTIVE OFFICER
```

(2) 第一次迭代，返回一个成员

```
MANAGERID  EMPLOYEEID  TITLE
---------  ----------  ---------------------------
1  273  VICE PRESIDENT OF SALES
```

(3) 第二次迭代，返回三个成员

```
MANAGERIDEMPLOYEEID  TITLE
---  -----------------------------  ------
273  16    MARKETING MANAGER
273  274  NORTH AMERICAN SALES MANAGER
273  285  PACIFIC SALES MANAGER
```

(4) 第三次迭代，返回四个成员

```
MANAGERID  EMPLOYEEID  TITLE
---------  ----------  -----------------------------
16  23  MARKETING SPECIALIST
274 275 SALES REPRESENTATIVE
274 276 SALES REPRESENTATIVE
285 286 SALES REPRESENTATIVE
```

(5) 第四次迭代，返回空集。递归结束。

(6) 正在运行的查询返回的最终结果集是定位点成员和递归成员生成的所有结果集的并集（UNION ALL）。

```
MANAGERID  EMPLOYEEID  TITLE
---------  ----------  -----------------------------
NULL    1       CHIEF EXECUTIVE OFFICER
1       273     VICE PRESIDENT OF SALES
273     16      MARKETING MANAGER
273     274     NORTH AMERICAN SALES MANAGER
273     285     PACIFIC SALES MANAGER
16      23      MARKETING SPECIALIST
274     275     SALES REPRESENTATIVE
274     276     SALES REPRESENTATIVE
285     286     SALES REPRESENTATIVE
```

4.5 合并查询结果

DM 提供了一种集合运算符：UNION，这种运算符将两个或多个查询块的结果集合并为一个结果集输出。语法如下：

语法格式

```
<查询表达式>
UNION [ALL][DISTINCT]
[ ( ]<查询表达式> [ ) ];
```

使用说明

1. 每个查询块的查询列数目必须相同；
2. 每个查询块对应的查询列的数据类型必须兼容；
3. 不允许查询列含有 BLOB、CLOB 或 IMAGE、TEXT 等大字段类型；
4. 在 UNION 后的可选项关键字 ALL 的意思是保持所有重复，而没有 ALL 的情况下表示删除所有重复；
5. 在 UNION 后的可选项关键字 DISTINCT 的意思是删除所有重复。缺省值为 DISTINCT。

举例说明

例 查询所有图书的出版商，查询所有图书供应商的名称，将两者连接，并去掉重复行。

```
SELECT PUBLISHER FROM PRODUCTION.PRODUCT
UNION
SELECT NAME FROM PURCHASING.VENDOR ORDER BY 1;
```

查询结果如表 4.5.1 所示。

表 4.5.1

PUBLISHER
21 世纪出版社
北京十月文艺出版社
长江文艺出版社
广州出版社
机械工业出版社
清华大学出版社 （注：末尾含空格）
清华大学出版社（注：末尾不含空格）
人民文学出版社
人民邮电出版社
上海出版社
上海画报出版社
外语教学与研究出版社
文学出版社
中华书局

例 UNION ALL。

```
SELECT  PUBLISHER  FROM  PRODUCTION.PRODUCT
UNION ALL
SELECT  NAME  FROM  PURCHASING.VENDOR ORDER BY 1;
```

查询结果如表 4.5.2 所示。

表 4.5.2

PUBLISHER
21 世纪出版社
21 世纪出版社
北京十月文艺出版社
长江文艺出版社

广州出版社
广州出版社
机械工业出版社
机械工业出版社
清华大学出版社
清华大学出版社
人民文学出版社
人民邮电出版社
上海出版社
上海出版社
上海画报出版社
外语教学与研究出版社
外语教学与研究出版社
文学出版社
中华书局
中华书局
中华书局

4.6 GROUP BY 和 HAVING 子句

GROUP BY 子句逻辑地将由 WHERE 子句返回的临时结果重新编组。结果是行的集合，一组内一个分组列的所有值都是相同的。HAVING 子句用于为组设置检索条件。

4.6.1 GROUP BY 子句的使用

GROUP BY 子句是 SELECT 语句的可选项部分。它定义了分组表。GROUP BY 子句语法如下：

```
<GROUP BY 子句> ::= GROUP BY <group_by 项>{,<group_by 项>}
<group_by 项>::=<分组项> | <ROLLUP 项> | <CUBE 项> | <GROUPING SETS 项>
<分组项>::= <值表达式>
<ROLLUP 项>::=ROLLUP (<分组项>)
<CUBE 项>::=CUBE (<分组项>)
<GROUPING SETS 项>::=GROUPING SETS(<GROUP 项>{,<GROUP 项>})
<GROUP 项>::=<分组项>
            |(<分组项>{,<分组项>})
            |()
```

GROUP BY 定义了分组表：行组的集合，其中每一个组由其中所有分组列的值都相等的行构成。

例 统计每个部门的员工数。

```
SELECT DEPARTMENTID,COUNT(*)  FROM  RESOURCES.EMPLOYEE_DEPARTMENT  GROUP BY  DEPARTMENTID;
```

查询结果如表 4.6.1 所示。

表 4.6.1

DEPARTMENTID	COUNT(*)
4	1
3	1
2	3
1	2

系统执行此语句时，首先将 EMPLOYEE_DEPARTMENT 表按 DEPARTMENTID 列进行分组，相同的 DEPARTMENTID 为一组，然后对每一组使用集函数 COUNT(*)，统计该组内的记录个数，如此继续，直到处理完最后一组，返回查询结果。

如果存在 WHERE 子句，系统先根据 WHERE 条件进行过滤，然后对满足条件的记录进行分组。

此外，GROUP BY 不会对结果集排序。如果需要排序，可以使用 ORDER BY 子句。

例 求小说类别包含的子类别所对应的产品数量，并按子类别编号的升序排列。

```
SELECT A1.PRODUCT_SUBCATEGORYID AS 子分类编号,A3.NAME AS 子分类名,count(*)AS 数量
FROM PRODUCTION.PRODUCT A1,
PRODUCTION.PRODUCT_CATEGORY A2,
PRODUCTION.PRODUCT_SUBCATEGORY A3
WHERE A1.PRODUCT_SUBCATEGORYID=A3.PRODUCT_SUBCATEGORYID
AND  A2.PRODUCT_CATEGORYID=A3.PRODUCT_CATEGORYID
AND  A2.NAME='小说'
GROUP  BY  A1.PRODUCT_SUBCATEGORYID,A3.NAME
ORDER  BY  A1.PRODUCT_SUBCATEGORYID;
```

查询结果如表 4.6.2 所示。

表 4.6.2

子分类编号	子分类名	数量
1	世界名著	1
2	武侠	1
4	四大名著	2

使用 GROUP BY 要注意以下问题：

1. 在 GROUP BY 子句中的每一列必须明确地命名属于在 FROM 子句中命名的表的一列。分组列的数据类型不能是多媒体数据类型；
2. 分组列不能为集函数表达式或者在 SELECT 子句中定义的别名；
3. 当分组列值包含空值时，则空值作为一个独立组；
4. 当分组列包含多个列名时，则按照 GROUP BY 子句中列出现的顺序进行分组；
5. GROUP BY 子句中至多可包含 255 个分组列；
6. ROLLUP\CUBE\GROUPING SETS 组合不能超过 9 个。

4.6.2 ROLLUP 的使用

ROLLUP 主要用于统计分析，对分组列以及分组列的部分子集进行分组，输出用户需要的结果。语法如下：

```
GROUP BY ROLLUP (<分组项>)
```

```
<分组项>::= <列名> | <值表达式>{,<列名> | <值表达式>}
```

假如 ROLLUP 分组列为(A, B, C)，首先对(A,B,C)进行分组，然后对(A,B)进行分组，接着对(A)进行分组，最后对全表进行查询，无分组列，其中查询项中出现在 ROLLUP 中的列设为 NULL。查询结果是把每种分组的结果集进行 UNION ALL 合并输出。如果分组列为 n 列，则共有 n+1 种组合方式。

例 按小区住址和所属行政区域统计员工居住分布情况。

```
SELECT CITY , ADDRESS1, COUNT(*) as NUMS FROM PERSON.ADDRESS GROUP BY
ROLLUP(CITY, ADDRESS1);
```

查询结果如表 4.6.3 所示。

表 4.6.3

CITY	ADDRESS1	NUMS
武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
武汉市洪山区	洪山区保利花园 50-1-304	1
武汉市洪山区	洪山区保利花园 51-1-702	1
武汉市洪山区	洪山区关山春晓 51-1-702	1
武汉市洪山区	洪山区关山春晓 55-1-202	1
武汉市洪山区'	洪山区关山春晓 10-1-202	1
武汉市洪山区	洪山区关山春晓 11-1-202	1
武汉市洪山区	洪山区光谷软件园 C1_501	1
武汉市青山区	青山区青翠苑 1 号	1
武汉市武昌区	武昌区武船新村 115 号	1
武汉市武昌区	武昌区武船新村 1 号	1
武汉市汉阳区	汉阳大道熊家湾 15 号	1
武汉市江汉区	江汉区发展大道 561 号	1
武汉市江汉区	江汉区发展大道 555 号	1
武汉市江汉区	江汉区发展大道 423 号	1
武汉市洪山区	NULL	9
武汉市青山区	NULL	1
武汉市武昌区	NULL	2
武汉市汉阳区	NULL	1
武汉市江汉区	NULL	3
NULL	NULL	16

上例中的查询等价于：

```
SELECT CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY
CITY, ADDRESS1
UNION ALL
SELECT CITY , NULL, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY CITY
UNION ALL
SELECT NULL , NULL, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY 0;
```

使用 ROLLUP 要注意以下事项：

1. ROLLUP 项不能包含集函数；

2. 不支持包含 ROWNUM、WITH FUNCTION 的相关查询；
3. 不支持包含存在 ROLLUP 的嵌套相关子查询；
4. ROLLUP 项最多支持 511 个；
5. ROLLUP 项不能引用外层列。

4.6.3 CUBE 的使用

CUBE 的使用场景与 ROLLUP 类似，常用于统计分析，对分组列以及分区列的所有子集进行分组，输出所有分组结果。语法如下：

```
GROUP BY CUBE (<分组项>)
<分组项>::= <列名> | <值表达式>{,<列名> | <值表达式>}
```

假如，CUBE 分组列为(A, B, C)，则首先对(A,B,C)进行分组，然后依次对(A,B)、(A,C)、(A)、(B,C)、(B)、(C)六种情况进行分组，最后对全表进行全表进行查询，无分组列，其中查询项存在于 CUBE 列表的列设置为 NULL。输出为每种分组的结果集进行 UNION ALL。CUBE 分组共有 2^n 种组合方式。CUBE 最多支持 9 列。

例 按小区住址、所属行政区域统计员工居住分布情况。

```
SELECT CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY CUBE(CITY, ADDRESS1);
```

查询结果如表 4.6.4 所示。

表 4.6.4

CITY	ADDRESS1	NUMS
武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
武汉市洪山区	洪山区保利花园 50-1-304	1
武汉市洪山区	洪山区保利花园 51-1-702	1
武汉市洪山区	洪山区关山春晓 51-1-702	1
武汉市洪山区	洪山区关山春晓 55-1-202	1
武汉市洪山区	洪山区关山春晓 10-1-202	1
武汉市洪山区	洪山区关山春晓 11-1-202	1
武汉市洪山区	洪山区光谷软件园 C1_501	1
武汉市青山区	青山区青翠苑 1 号	1
武汉市武昌区	武昌区武船新村 115 号	1
武汉市武昌区	武昌区武船新村 1 号	1
武汉市汉阳区	汉阳大道熊家湾 15 号	1
武汉市江汉区	江汉区发展大道 561 号	1
武汉市江汉区	江汉区发展大道 555 号	1
武汉市江汉区	江汉区发展大道 423 号	1
武汉市洪山区	NULL	9
武汉市青山区	NULL	1
武汉市武昌区	NULL	2
武汉市汉阳区	NULL	1
武汉市江汉区	NULL	3
NULL	洪山区 369 号金地太阳城 56-1-202	1

NULL	洪山区 369 号金地太阳城 57-2-302	1
NULL	洪山区保利花园 50-1-304	1
NULL	洪山区保利花园 51-1-702	1
NULL	洪山区关山春晓 51-1-702	1
NULL	洪山区关山春晓 55-1-202	1
NULL	洪山区关山春晓 10-1-202	1
NULL	洪山区关山春晓 11-1-202	1
NULL	洪山区光谷软件园 C1_501	1
NULL	青山区青翠苑 1 号	1
NULL	武昌区武船新村 115 号	1
NULL	武昌区武船新村 1 号	1
NULL	汉阳大道熊家湾 15 号	1
NULL	江汉区发展大道 561 号	1
NULL	江汉区发展大道 555 号	1
NULL	江汉区发展大道 423 号	1
NULL	NULL	16

上例中的查询等价于：

```
SELECT CITY , ADDRESS1, COUNT(*) AS NUMS  FROM  PERSON.ADDRESS  GROUP  BY
CITY, ADDRESS1
UNION ALL
SELECT CITY , NULL, COUNT(*) AS NUMS  FROM  PERSON.ADDRESS  GROUP  BY  CITY
UNION ALL
SELECT NULL , ADDRESS1, COUNT(*) AS NUMS  FROM  PERSON.ADDRESS  GROUP  BY
ADDRESS1
UNION ALL
SELECT NULL , NULL, COUNT(*) AS NUMS  FROM  PERSON.ADDRESS;
```

使用 CUBE 要注意以下事项：

1． CUBE 项不能包含集函数；
2． 不支持包含 WITH FUNCTION 的相关查询；
3． 不支持包含存在 CUBE 的嵌套相关子查询；
4． CUBE 项最多支持 9 个；
5． CUBE 项不能引用外层列。

4.6.4 GROUPING 的使用

GROUPING 可以视为集函数，一般用于含 GROUP BY 的语句中，标识某子结果集是否是按指定分组项分组的结果，如果是，GROUPING 值为 0；否则为 1。语法如下：

```
<GROUPING 项>::=GROUPING (<分组项>)
<分组项>::= <列名> | <值表达式>
```

使用约束说明：

1. GROUPING 中只能包含一列；

2．GROUPING 只能在 GROUP BY 查询中使用；

3．GROUPING 不能在 WHERE 或连接条件中使用；

4．GROUPING 支持表达式运算。例如 GROUPING(c1) + GROUPING(c2)。

例 按小区住址和所属行政区域统计员工居住分布情况。

```
SELECT GROUPING(CITY) AS G_CITY,GROUPING(ADDRESS1) AS G_ADD, CITY , ADDRESS1,
COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY ROLLUP(CITY, ADDRESS1);
```

查询结果如表 4.6.5 所示。

表 4.6.5

G_CITY	G_ADD	CITY	ADDRESS1	NUMS
0	0	武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
0	0	武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
0	0	武汉市洪山区	洪山区保利花园 50-1-304	1
0	0	武汉市洪山区	洪山区保利花园 51-1-702	1
0	0	武汉市洪山区	洪山区关山春晓 51-1-702	1
0	0	武汉市洪山区	洪山区关山春晓 55-1-202	1
0	0	武汉市洪山区'	洪山区关山春晓 10-1-202	1
0	0	武汉市洪山区	洪山区关山春晓 11-1-202	1
0	0	武汉市洪山区	洪山区光谷软件园 C1_501	1
0	0	武汉市青山区	青山区青翠苑 1 号	1
0	0	武汉市武昌区	武昌区武船新村 115 号	1
0	0	武汉市武昌区	武昌区武船新村 1 号	1
0	0	武汉市汉阳区	汉阳大道熊家湾 15 号	1
0	0	武汉市江汉区	江汉区发展大道 561 号	1
0	0	武汉市江汉区	江汉区发展大道 555 号	1
0	0	武汉市江汉区	江汉区发展大道 423 号	1
0	1	武汉市洪山区	NULL	9
0	1	武汉市青山区	NULL	1
0	1	武汉市武昌区	NULL	2
0	1	武汉市汉阳区	NULL	1
0	1	武汉市江汉区	NULL	3
1	1	NULL	NULL	16

4.6.5 GROUPING SETS 的使用

GROUPING SETS 是对 GROUP BY 的扩展，可以指定不同的列进行分组，每个分组列集作为一个分组单元。使用 GROUPING SETS，用户可以灵活的指定分组方式，避免 ROLLUP/CUBE 过多的分组情况，满足实际应用需求。

GROUPING SETS 的分组过程为依次按照每一个分组单元进行分组，最后把每个分组结果进行 UNION ALL 输出最终结果。如果查询项不属于分组列，则用 NULL 代替。语法如下：

```
GROUP BY GROUPING SETS (<分组项>)
<分组项> ::= <分组子项> {,<分组子项>}
<分组子项> ::= <表达式> | () |(<表达式>{,<表达式>})
<表达式> ::= <列名> | <值表达式>
```

例 按照邮编、住址和行政区域统计员工住址分布情况。

```
SELECT CITY , ADDRESS1, POSTALCODE, COUNT(*) AS NUMS  FROM  PERSON.ADDRESS
GROUP  BY  GROUPING SETS((CITY, ADDRESS1), POSTALCODE);
```

查询结果如表 4.6.6 所示。

表 4.6.6

CITY	ADDRESS1	POSTALCODE	NUMS
武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	NULL	1
武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	NULL	1
武汉市洪山区	洪山区保利花园 50-1-304	NULL	1
武汉市洪山区	洪山区保利花园 51-1-702	NULL	1
武汉市洪山区	洪山区关山春晓 51-1-702	NULL	1
武汉市洪山区	洪山区关山春晓 55-1-202	NULL	1
武汉市洪山区'	洪山区关山春晓 10-1-202	NULL	1
武汉市洪山区	洪山区关山春晓 11-1-202	NULL	1
武汉市洪山区	洪山区光谷软件园 C1_501	NULL	1
武汉市青山区	青山区青翠苑 1 号	NULL	1
武汉市武昌区	武昌区武船新村 115 号	NULL	1
武汉市武昌区	武昌区武船新村 1 号	NULL	1
武汉市汉阳区	汉阳大道熊家湾 15 号	NULL	1
武汉市江汉区	江汉区发展大道 561 号	NULL	1
武汉市江汉区	江汉区发展大道 555 号	NULL	1
武汉市江汉区	江汉区发展大道 423 号	NULL	1
NULL	NULL	430073	9
NULL	NULL	430080	1
NULL	NULL	430063	2
NULL	NULL	430050	1
NULL	NULL	430023	3

上例中的查询等价于：

```
SELECT CITY , ADDRESS1, NULL , COUNT(*) AS NUMS  FROM  PERSON.ADDRESS  GROUP
BY  CITY, ADDRESS1
UNION ALL
SELECT NULL  ,  NULL,  POSTALCODE  ,COUNT(*)  AS  NUMS   FROM   PERSON.ADDRESS
GROUP  BY  POSTALCODE;
```

使用 GROUPING SETS 要注意以下事项：

1. GROUPING SETS 项不能包含集函数；
2. 不支持包含 WITH FUNCTION 的相关查询；
3. 不支持包含存在 GROUPING SETS 的嵌套相关子查询；
4. GROUPING SETS 项最多支持 512 个；
5. GROUPING SETS 项不能引用外层列。

4.6.6 GROUPING_ID 的使用

GROUPING_ID 表示参数列是否为分组列。返回值的每一个二进制位表示对应的参数列是否为分组列，如果是分组列，该位值为 0；否则为 1。

使用 GROUPING_ID 可以按照列的分组情况过滤结果集。

语法如下：

```
<GROUPING_ID 项>::=GROUPING_ID (<分组项>{,<分组项>)
<分组项>::= <列名> | <值表达式>
```

使用约束说明：

1. GROUPING_ID 中至少包含一列，最多包含 63 列；
2. GROUPING_ID 只能与分组项一起使用；
3. GROUPING_ID 支持表达式运算；
4. GROUPING_ID 作为分组函数，不能出现在 where 或连接条件中。

例 按小区住址和所属行政区域统计员工居住分布情况。

```
SELECT GROUPING(CITY) AS G_CITY,GROUPING(ADDRESS1) AS G_ADD, GROUPING_ID(CITY, ADDRESS1) AS G_CA,CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY ROLLUP(CITY, ADDRESS1);
```

查询结果如表 4.6.7 所示。

表 4.6.7

G_CITY	G_ADD	G_CA	CITY	ADDRESS1	NUMS
0	0	0	武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
0	0	0	武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
0	0	0	武汉市洪山区	洪山区保利花园 50-1-304	1
0	0	0	武汉市洪山区	洪山区保利花园 51-1-702	1
0	0	0	武汉市洪山区	洪山区关山春晓 51-1-702	1
0	0	0	武汉市洪山区	洪山区关山春晓 55-1-202	1
0	0	0	武汉市洪山区'	洪山区关山春晓 10-1-202	1
0	0	0	武汉市洪山区	洪山区关山春晓 11-1-202	1
0	0	0	武汉市洪山区	洪山区光谷软件园 C1_501	1
0	0	0	武汉市青山区	青山区青翠苑 1 号	1
0	0	0	武汉市武昌区	武昌区武船新村 115 号	1
0	0	0	武汉市武昌区	武昌区武船新村 1 号	1
0	0	0	武汉市汉阳区	汉阳大道熊家湾 15 号	1
0	0	0	武汉市江汉区	江汉区发展大道 561 号	1
0	0	0	武汉市江汉区	江汉区发展大道 555 号	1
0	0	0	武汉市江汉区	江汉区发展大道 423 号	1
0	1	1	武汉市洪山区	NULL	9
0	1	1	武汉市青山区	NULL	1
0	1	1	武汉市武昌区	NULL	2
0	1	1	武汉市汉阳区	NULL	1
0	1	1	武汉市江汉区	NULL	3
1	1	3	NULL	NULL	16

4.6.7 GROUP_ID 的使用

GROUP_ID 表示结果集来自于哪一个分组，用于区别相同分组的结果集。如果有 N 个相同分组，则 GROUP_ID 取值从 0..N-1。每组的初始值为 0。

当查询包含多个分组时，使用 GROUP_ID 可以方便的过滤相同分组的结果集。

```
<GROUP_ID 项>::=GROUP_ID()
```

使用约束说明：

1. GROUP_ID 不包含参数；
2. GROUP_ID 只能与分组项一起使用；
3. GROUP_ID 支持表达式运算；
4. GROUP_ID 作为分组函数，不能出现在 WHERE 或连接条件中。

例 按小区住址和所属行政区域统计员工居住分布情况。

```
SELECT GROUPING(CITY) AS G_CITY,GROUPING(ADDRESS1) AS G_ADD, GROUP_ID() AS GID,CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY ROLLUP(CITY, ADDRESS1), CITY;
```

查询结果如表 4.6.8 所示。

表 4.6.8

G_CITY	G_ADD	GID	CITY	ADDRESS1	NUMS
0	0	0	武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
0	0	0	武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
0	0	0	武汉市洪山区	洪山区保利花园 50-1-304	1
0	0	0	武汉市洪山区	洪山区保利花园 51-1-702	1
0	0	0	武汉市洪山区	洪山区关山春晓 51-1-702	1
0	0	0	武汉市洪山区	洪山区关山春晓 55-1-202	1
0	0	0	武汉市洪山区'	洪山区关山春晓 10-1-202	1
0	0	0	武汉市洪山区	洪山区关山春晓 11-1-202	1
0	0	0	武汉市洪山区	洪山区光谷软件园 C1_501	1
0	0	0	武汉市青山区	青山区青翠苑 1 号	1
0	0	0	武汉市武昌区	武昌区武船新村 115 号	1
0	0	0	武汉市武昌区	武昌区武船新村 1 号	1
0	0	0	武汉市汉阳区	汉阳大道熊家湾 15 号	1
0	0	0	武汉市江汉区	江汉区发展大道 561 号	1
0	0	0	武汉市江汉区	江汉区发展大道 555 号	1
0	0	0	武汉市江汉区	江汉区发展大道 423 号	1
0	1	0	武汉市洪山区	NULL	9
0	1	0	武汉市青山区	NULL	1
0	1	0	武汉市武昌区	NULL	2
0	1	0	武汉市汉阳区	NULL	1
0	1	0	武汉市江汉区	NULL	3
0	1	1	武汉市洪山区	NULL	9
0	1	1	武汉市青山区	NULL	1
0	1	1	武汉市武昌区	NULL	2
0	1	1	武汉市汉阳区	NULL	1

0	1	1	武汉市江汉区	NULL	3

4.6.8 HAVING 子句的使用

HAVING 子句是 SELECT 语句的可选项部分，它也定义了一个成组表。HAVING 子句语法如下：

```
<HAVING 子句> ::= HAVING <搜索条件>
<搜索条件>::= <表达式>
```

HAVING 子句定义了一个成组表，其中只含有搜索条件为 TRUE 的那些组，且通常跟随一个 GROUP BY 子句。HAVING 子句与组的关系正如 WHERE 子句与表中行的关系。WHERE 子句用于选择表中满足条件的行，而 HAVING 子句用于选择满足条件的组。

例 统计出同一子类别的产品数量大于 1 的子类别名称，数量，并按数量从小到大的顺序排列。

```
SELECT A2.NAME AS 子分类名, COUNT (*)AS 数量
FROM PRODUCTION.PRODUCT A1,
PRODUCTION.PRODUCT_SUBCATEGORY A2
WHERE A1.PRODUCT_SUBCATEGORYID=A2.PRODUCT_SUBCATEGORYID
GROUP  BY  A2.NAME
HAVING COUNT(*)>1
ORDER  BY  2;
```

查询结果如表 4.6.9 所示。

表 4.6.9

子分类名	数量
四大名著	2

系统执行此语句时，首先将 PRODUCT 表和 PRODUCT_SUBCATEGORY 表中的各行按相同的 SUBCATEGORYID 作连接，再按子类别名的取值进行分组，相同的子类别名为一组，然后对每一组使用集函数 COUNT(*)，统计该组内产品的数量，如此继续，直到最后一组。再选择产品数量大于 1 的组作为查询结果。

4.7 ORDER BY 子句

ORDER BY 子句可以选择性地出现在<查询表达式>之后，它规定了当行由查询返回时应具有的顺序。ORDER BY 子句的语法如下：

```
<ORDER BY 子句> ::= ORDER [SIBLINGS] BY < order_by_list>
< order_by_list>::= < order_by_list>,]<order_by_item>
<order_by_item>::=  <exp> [ASC | DESC] [NULLS FIRST|LAST]
< exp >::=<无符号整数> | <列说明> | <值表达式>
```

使用说明

1. ORDER BY 子句提供了要排序的项目清单和他们的排序顺序：递增顺序(ASC，默认)或是递减顺序(DESC)。它必须跟随在<查询表达式>之后，因为它是在查询计算得出的最终结果上进行操作的；

2. 排序键可以是任何在查询清单中的列的名称，或者是对最终结果表的列计算的表达

式(即使这一列不在选择清单中)，也可以是子查询。对于 UNION 查询语句，排序键必须在第一个查询子句中出现；对于 GROUP BY 分组的排序，排序键可以使用集函数，但 GROUP BY 分组中必须包含查询列中所有列；

3. <无符号整数> 对应 SELECT 后结果列的序号。当用<无符号整数>代替列名时，<无符号整数>不应大于 SELECT 后结果列的个数。如下面例子中 ORDER BY 3，因查询结果列只有 2 列，无法进行排序，系统将会报错。若采用其他常量表达式(如：-1，3×6)作为排序列，将不影响最终结果表的行输出顺序；

4. SIBLINGS 关键字必须与 CONNECT BY 一起配合使用，专门用于指定层次查询中相同层次数据返回的顺序。详见 4.12.5 层次查询层内排序；

5. 无论采用何种方式标识想要排序的结果列，它们都不支持多媒体数据类型(如 IMAGE、TEXT、BLOB 和 CLOB)；

6. 当排序列值包含 NULL 时，根据指定的“NULLS FIRST|LAST”决定包含空值的行是排在最前还是最后，缺省为 NULLS FIRST；

7. 当排序列包含多个列名时，系统则按列名从左到右排列的顺序，先按左边列将查询结果排序，当左边排序列值相等时，再按右边排序列排序……如此右推，逐个检查调整，最后得到排序结果；

8. 由于 ORDER BY 只能在最终结果上操作，不能将其放在查询中；

9. 如果 ORDER BY 后面使用集函数，则必须使用 GROUP BY 分组，且 GROUP BY 分组中必须包含查询列中所有列；

10. ORDER BY 子句中至多可包含 255 个排序列。

例 将 RESOURCES.DEPARTMENT 表中的资产总值按从大到小的顺序排列。

```
SELECT * FROM RESOURCES.DEPARTMENT ORDER BY DEPARTMENTID DESC;
```

等价于：

```
SELECT * FROM RESOURCES.DEPARTMENT ORDER BY 1 DESC;
```

查询结果如表 4.7.1 所示。

表 4.7.1

DEPARTMENTID	NAME
5	广告部
4	行政部门
3	人力资源
2	销售部门
1	采购部门

例

```
SELECT * FROM RESOURCES.DEPARTMENT ORDER BY 3;
```

系统报错：无效的 ORDER BY 语句。

4.8 FOR UPDATE 子句

FOR UPDATE 子句可以选择性地出现在<查询表达式>之后。普通 SELECT 查询不会修改行数据物理记录上的 TID 事务号，FOR UPDATE 会修改行数据物理记录上的 TID 事务号并对该 TID 上锁，以保证该更新操作的待更新数据不被其他事务修改。

语法格式

```
<FOR UPDATE 子句> ::= FOR READ ONLY
```

```
                        | <FOR UPDATE 选项>
    <FOR UPDATE 选项> ::= FOR UPDATE [OF <选择列表>] [ NOWAIT
                                                      |WAIT N
                                                      |[N]SKIP LOCKED
                                                    ]
    <选择列表> :: = [<模式名>.] <基表名>|<视图名> .] <列名> {,[<模式名>.] <基表名>|<
视图名> .] <列名>}
```

参数

1. FOR READ ONLY 表示查询不可更新；

2. OF <选择列表>指定待更新表的列。指定某张表的列，即为锁定某张表。游标更新时，仅能更新指定的列；

3. NOWAIT，WAIT，SKIP LOCKED 等子句表示当试图上锁的行数据 TID 已经被其他事务上锁的处理方式：

   1) NOWAIT 表示不等待，直接报错返回；
   2) WAIT N 表示等待一段时间，其中的 N 值由用户指定，单位为秒。等待成功继续上锁，失败则报错返回。WAIT 的指定值必须大于 0，如果设置 0 自动转成 NOWAIT 方式；
   3) [N] SKIP LOCKED 表示上锁时跳过已经被其他事务锁住的行，不返回这些行给客户端。N 是整数，为 DM 特有的语法，表示当取得了 N 条数据后，便不再取数据了，直接返回 N 条结果；
   4) 如果 FOR UPDATE 不设置以上三种子句，则会一直等待锁被其他事务释放；
   5) INI 参数 LOCK_TID_MODE 用来标记 SELECT FOR UPDATE 封锁方式。0 表示结果集小于 100 行时，直接封锁 TID，超过 100 行升级为表锁。1 表示不升级表锁，一律使用 TID 锁。默认为 1。

例 查询 RESOURCES.DEPARTMENT 表中的资产。

```
    SELECT  *  FROM  RESOURCES.DEPARTMENT FOR UPDATE; --只要 FOR UPDATE 语句不提交，
其他会话就不能修改此结果集。
```

查询结果如表 4.8.1 所示。

表 4.8.1

DEPARTMENTID	NAME
5	广告部
4	行政部门
3	人力资源
2	销售部门
1	采购部门

需要说明的是：

1. 以下情况 SELECT FOR UPDATE 查询会报错：

   1) 带 GROUP BY 的查询，如 SELECT C1, COUNT(C2) FROM TEST GROUP BY C1 FOR UPDATE;
   2) 带聚集函数的查询，如 SELECT MAX(C1)FROM TEST FOR UPDATE;
   3) 带分析函数的查询，如 SELECT MAX(C1) OVER(PARTITION BY C1) FROM TEST FOR UPDATE;
   4) 对以下表类型的查询：外部表、物化视图、位图连接索引表（内部辅助表，命名为 BMJ$_索引名）、位图索引表（内部辅助表，命名为 BM$_基表名）、系统

表和 HUGE 表；

5) WITH 子句，如 WITH TEST(C1) AS (SELECT C1 FROM T FOR UPDATE)SELECT * FROM TEST。

2. 涉及 DBLINK 的 SELECT FOR UPDATE 查询仅支持单表；

3. 如果结果集中包含 LOB 对象，会再封锁 LOB 对象；

4. 支持多表连接的情况，会封锁涉及到的所有表的行数据；

5. 多表连接的时候，如果用 OF <选择列表>指定具体列，只会检测和封锁对应的表。例如 SELECT C1 FROM TEST, TESTB FOR UPDATE OF TEST.C1 即使 TESTB 表类型不支持 FOR UPDATE，上述语句还是可以成功。

4.9 TOP 子句

在 DM 中，可以使用 TOP 子句来筛选结果。语法如下：

```
<TOP 子句>::=TOP <n>
                | <<n>,<n>>
                | <n> PERCENT
                | <n> WITH TIES
                | <n> PERCENT WITH TIES
<n>::=整数（>=0）
```

参数

1. TOP <n> 选择结果的前 n 条记录；

2. TOP <n1>,<n2> 选择第 n1 条记录之后的 n2 条记录；

3. TOP <n> PERCENT 表示选择结果的前 n%条记录；

4. TOP <n> PERCENT WITH TIES 表示选择结果的前 n%条记录，同时指定结果集可以返回额外的行。额外的行是指与最后一行以相同的排序键排序的所有行。WITH TIES 必须与 ORDER BY 子句同时出现，如果没有 ORDER BY 子句，则忽略 WITH TIES。

例 查询现价最贵的两种产品的编号和名称。

```
SELECT TOP 2 PRODUCTID,NAME FROM PRODUCTION.PRODUCT
ORDER BY NOWPRICE DESC;
```

其结果如表 4.9.1 所示：

表 4.9.1

PRODUCTID	NAME
10	噼里啪啦丛书(全 7 册)
6	长征

例 查询现价第二贵的产品的编号和名称。

```
SELECT TOP 1,1 PRODUCTID,NAME FROM PRODUCTION.PRODUCT
ORDER BY NOWPRICE DESC;
```

其结果如表 4.9.2 所示：

表 4.9.2

PRODUCTID	NAME
6	长征

例 查询最新出版日期的70%的产品编号、名称和出版日期。

```
SELECT TOP 70 PERCENT WITH TIES PRODUCTID,NAME,PUBLISHTIME FROM PRODUCTION.PRODUCT ORDER BY PUBLISHTIME DESC;
```

其结果如表 4.9.3 所示：

表 4.9.3

PRODUCTID	NAME	PUBLISHTIME
7	数据结构(C 语言版)(附光盘)	2007-03-01
5	鲁迅文集(小说、散文、杂文)全两册	2006-09-01
6	长征	2006-09-01
3	老人与海	2006-08-01
8	工作中无小事	2006-01-01
4	射雕英雄传(全四册)	2005-12-01
2	水浒传	2005-04-01
1	红楼梦	2005-04-01

4.10 LIMIT 限定条件

在 DM 中，可以使用限定条件对结果集做出筛选，支持 LIMIT 子句和 ROW_LIMIT 子句两种方式。

4.10.1 LIMIT 子句

LIMIT 子句按顺序选取结果集中某条记录开始的 N 条记录。语法如下

```
<LIMIT 子句>::= LIMIT<  <记录数>
                    | <<记录数>,<记录数>>
                    | <<记录数> OFFSET <偏移量>>
                 >
                 | OFFSET <偏移量> LIMIT <记录数>
<记录数>::=<整数>
<偏移量>::=<整数>
```

共支持四种方式：

1．LIMIT N：选择前 N 条记录；

2．LIMIT M,N：选择第 M 条记录之后的 N 条记录；

3．LIMIT M OFFSET N：选择第 N 条记录之后的 M 条记录；

4．OFFSET N LIMIT M：选择第 N 条记录之后的 M 条记录。

注意：LIMIT 不能与 TOP 同时出现在查询语句中。

例 查询前 2 条记录

```
SELECT PRODUCTID , NAME FROM PRODUCTION.PRODUCT LIMIT 2;
```

其结果如表 4.10.1 所示：

表 4.10.1

PRODUCTID	NAME
1	红楼梦
2	水浒传

例 查询第 3，4 个登记的产品的编号和名称。

```
SELECT PRODUCTID, NAME  FROM  PRODUCTION.PRODUCT  LIMIT  2  OFFSET  2;
```

其结果如表 4.10.2 所示：

表 4.10.2

PRODUCTID	NAME
3	老人与海
4	射雕英雄传(全四册)

例 查询前第 5，6，7 个登记的姓名。

```
SELECT PERSONID,NAME FROM PERSON.PERSON LIMIT 4,3;
```

其结果如表 4.10.3 所示：

表 4.10.3

PERSONID	NAME
5	孙丽
6	黄非
7	王菲

4.10.2 ROW_LIMIT 子句

ROW_LIMIT 子句用于指定查询结果中偏移位置的行数或者百分比行数，以便更为灵活地获取查询结果。

语法如下：

```
< ROW_LIMIT 子句>::= [OFFSET <offset> <ROW | ROWS> ] [<FETCH 说明>]
<FETCH 说明>::= FETCH <FIRST | NEXT> <大小> [PERCENT] < ROW | ROWS ><ONLY| WITH TIES>
```

参数

1．<offset> 指定查询返回行的起始偏移。必须为数字。offset 为负数时视为 0；为 NULL 或大于等于所返回的行数时，返回 0 行；为小数时，小数部分截断；

2．<FIRST | NEXT> FIRST 为从偏移为 0 的位置开始。 NEXT 为从指定的偏移的下一行开始获取结果。 只做注释说明的作用，没有实际的限定作用；

3．<大小>[PERCENT] 指定返回行的行数（无 PERCENT）或者百分比（有 PERCENT）。其中<大小>只能为数字。percent 指定为负数时，视为 0%；为 NULL 时返回 0 行，如果没有指定 percent，返回 1 行；

4．<ONLY | WITH TIES> 指定结果集是否返回额外的行。额外的行是指与最后一行以相同的排序键排序的所有行。ONLY 为只返回指定的行数。WITH TIES 必须与 ORDER BY 子句同时出现，如果没有 ORDER BY 子句，则忽略 WITH TIES。

使用说明

1．ROW_LIMIT 子句不能与 FOR UPDATE 子句一起使用；

2．使用 ROW_LIMIT 子句时，查询列中不能包含有 CURRVAL 或者 NEXTVAL 伪列；

3．视图的查询定义中包含有 ROW_LIMIT 子句时，这个视图不会增量刷新。

例 查询价格最便宜的 50%的商品

```
SELECT NAME, NOWPRICE FROM PRODUCTION.PRODUCT ORDER BY NOWPRICE FETCH FIRST 50 PERCENT ROWS ONLY;
```

其结果如表 4.10.4 所示：

表 4.10.4

NAME	NOWPRICE
老人与海	6.1000
突破英文基础词汇	11.100
工作中无小事	11.4000
水浒传	14.3000
红楼梦	15.2000

例 查询价格第 3 便宜开始的 3 条记录

```
SELECT NAME, NOWPRICE FROM PRODUCTION.PRODUCT ORDER BY NOWPRICE OFFSET 2 ROWS FETCH FIRST 3 ROWS ONLY;
```

其结果如表 4.10.5 所示：

表 4.10.5

NAME	NOWPRICE
工作中无小事	11.4000
水浒传	14.3000
红楼梦	15.2000

4.11 全文检索

DM 数据库提供多文本数据检索服务，包括全文索引和全文检索。全文索引为在字符串数据中进行复杂的词搜索提供了有效支持。全文索引存储关于词和词在特定列中的位置信息，全文检索利用这些信息，可以快速搜索包含某个词或某一组词的记录。

执行全文检索涉及到以下这些任务：

1．对需要进行全文检索的表和列进行注册；

2．对注册了的列的数据建立全文索引；

3．对注册了的列查询填充后的全文索引。

执行全文检索步骤如下：

1．建立全文索引；

2．修改（填充）全文索引；

3．使用带 CONTAINS 谓词的查询语句进行全文检索；

4．当数据表的全文索引列数据发生变化，则需要进行增量或者完全填充全文索引，以便可以查询到更新后的数据；

5．若不再需要全文索引，可以删除该索引；

6．在全文索引定义并填充后，才可进行全文检索。

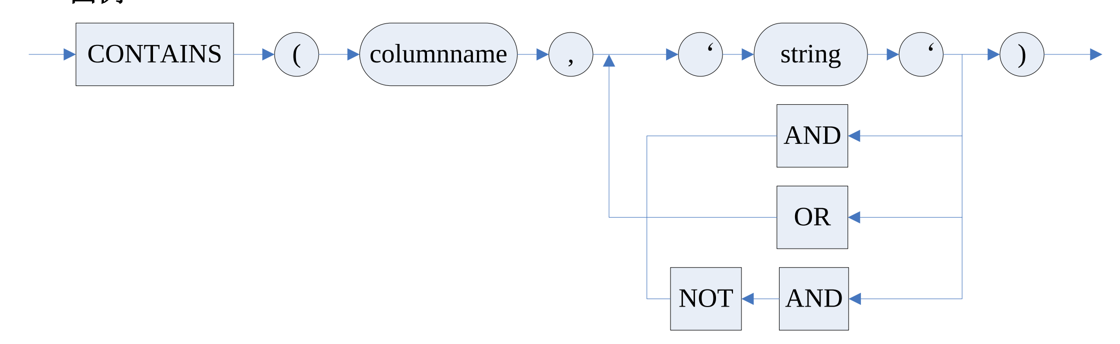
全文检索通过在查询语句中使用 CONTAINS 子句进行。

语法格式

```
CONTAINS ( <列名> , <检索条件> )
<检索条件>::= <布尔项> | <检索条件> <AND | OR | AND NOT> <布尔项>
```

```
<布尔项>::= '字符串'
```

图例



使用说明

1. 使用 CONTAINS 子句查询时，<列名>必须是已经建立了全文索引并填充后的列，否则系统会报错；

2. 支持精确字、词、短语及一段文字的查询，CONTAINS 谓词内支持 AND | AND NOT | OR 的使用，AND 的优先级高于 OR 的优先级；

3. 支持对每个精确词（单字节语言中没有空格或标点符号的一个或多个字符）或短语（单字节语言中由空格和可选的标点符号分隔的一个或多个连续的词）的匹配。对词或短语中字符的搜索不区分大小写；

4. 对于短语或一段文字的查询，根据词库，单个查找串被分解为若干个关键词，忽略词库中没有的词和标点符号，在索引上进行（关键词 AND 关键词）匹配查找。因而，不一定是精确查询；

5. 英文查询不区分大小写和全角半角中英文字符；

6. 不提供 Noise 文件，即不考虑忽略词或干扰词；

7. 不支持通配符"*"；

8. 不提供对模糊词或变形词的查找；

9. 不支持对结果集的相关度排名；

10. 检索条件子句可以和其他子句共同组成 WHERE 的检索条件。

举例说明

例 全文检索综合实例，以 PRODUCT 表为例。

(1)在 DESCRIPTION 列上定义全文索引。

```
CREATE CONTEXT INDEX INDEX1 ON PRODUCTION.PRODUCT(DESCRIPTION) LEXER CHINESE_VGRAM_LEXER;
```

(2)完全填充全文索引。

```
ALTER CONTEXT INDEX INDEX1 ON PRODUCTION.PRODUCT REBUILD;
```

(3)进行全文检索，查找描述里有“语言”字样的产品的编号和名称。

```
SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION, '语言');
```

结果如表 4.11.1 所示。

表 4.11.1

PRODUCTID	NAME
2	水浒传
7	数据结构(C 语言版)(附光盘)

(4) 进行全文检索，查找描述里有“语言”及“中国”字样的产品的编号和名称。

```
SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION,'
```

```
语言' AND '中国');
```

结果如表 4.11.2 所示。

表 4.11.2

PRODUCTID	NAME
2	水浒传

(5)进行全文检索，查找描述里有“语言”或“中国”字样的产品的编号和名称。

```
    SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION,'
语言' OR '中国');
```

结果如表 4.11.3 所示。

表 4.11.3

PRODUCTID	NAME
1	红楼梦
2	水浒传
7	数据结构(C 语言版)(附光盘)

(6)进行全文检索，查找描述里无“中国”字样的雇员的产品的编号和名称。

```
    SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE NOT CONTAINS(DESCRIPTION,
'中国');
```

结果如表 4.11.4 所示。

表 4.11.4

PRODUCTID	NAME
3	老人与海
4	射雕英雄传(全四册)
5	鲁迅文集(小说、散文、杂文)全两册
6	长征
7	数据结构(C 语言版)(附光盘)
8	工作中无小事
9	突破英文基础词汇
10	噼里啪啦丛书(全 7 册)

(7)进行全文检索，查找描述里有“C 语言”字样的产品的编号和名称。

```
    SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION,'C
语言');
```

结果如表 4.11.5 所示。

表 4.11.5

PRODUCTID	NAME
7	数据结构(C 语言版)(附光盘)

(8)对不再需要的全文索引进行删除。

```
    DROP CONTEXT INDEX INDEX1 ON PRODUCTION.PRODUCT;
```

4.12 层次查询子句

可通过层次查询子句进行层次查询，得到数据间的层次关系。在使用层次查询子句时，

可以使用层次查询相关的伪列、函数或操作符来明确层次查询结果中的相应层次信息。

4.12.1 层次查询子句

语法格式

```
<层次查询子句> ::=
        CONNECT BY [NOCYCLE]  <连接条件> [ START WITH <起始条件> ] |
        START WITH <起始条件> CONNECT BY [NOCYCLE] <连接条件>
<连接条件>::= <逻辑表达式>
<起始条件>::= <逻辑表达式>
```

参数

1. <连接条件>　逻辑表达式，指明层次数据间的层次连接关系；
2. <起始条件>　逻辑表达式，指明选择层次数据根数据的条件；
3. NOCYCLE 关键字用于指定数据导致环的处理方式，如果在层次查询子句中指定 NOCYCLE 关键字，会忽略导致环元组的儿子数据。否则，返回错误。

4.12.2 层次查询相关伪列

在使用层次查询子句时，可以通过相关的伪列来明确数据的层次信息。层次查询相关的伪列有：

1. LEVEL

该伪列表示当前元组在层次数据形成的树结构中的层数。LEVEL 的初始值为 1，即层次数据的根节点数据的 LEVEL 值为 1，之后其子孙节点的 LEVEL 依次递增。

2. CONNECT_BY_ISLEAF

该伪列表示当前元组在层次数据形成的树结构中是否是叶节点(即该元组根据连接条件不存在子结点)。是叶节点时为 1，否则为 0。

3. CONNECT_BY_ISCYCLE

该伪列表示当前元组是否会将层次数据形成环，该伪列只有在层次查询子句中表明 NOCYCLE 关键字时才有意义。如果元组的存在会导致层次数据形成环，该伪列值为 1，否则为 0。

4.12.3 层次查询相关操作符

1. PRIOR

PRIOR 操作符主要使用在层次查询子句中，指明 PRIOR 之后的参数为逻辑表达式中的父节点。

PRIOR 操作符还可以出现在查询项、WHERE 条件、GROUP BY 子句、集函数参数中，表示父层记录对应的值。

例：

```
    SELECT HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT CONNECT BY NOCYCLE  PRIOR
DEP_NAME = HIGH_DEP;  --DEP_NAME 为父节点。下一条记录的 HIGH_DEP 等于前一条记录的
DEP_NAME
```

```
    或者
    SELECT HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT CONNECT BY NOCYCLE  DEP_NAME
= PRIOR  HIGH_DEP;   -- HIGH_DEP 为父节点。下一条记录的 DEP_NAME 等于前一条记录的
HIGH_DEP
```

2. CONNECT_BY_ROOT

该操作符作为查询项，查询在层次查询结果中根节点的某列的值。

4.12.4 层次查询相关函数

语法格式

```
SYS_CONNECT_BY_PATH(col_name,char)
```

语句功能

层次查询。

使用说明

该函数得到从根节点到当前节点路径上所有节点名为 col_name 的某列的值，之间用 char 指明的字符分隔开。

4.12.5 层次查询层内排序

语法格式

```
ORDER SIBLINGS BY <order_by_list>
<order_by_list>请参考 4.7 ORDER BY 子句
```

语句功能

层次查询。

使用说明

ORDER SIBLINGS BY 用于指定层次查询中相同层次数据返回的顺序。在层次查询中使用 ORDER SIBLINGS BY，必须与 CONNECT BY 一起配合使用。但是，ORDER SIBLINGS BY 不能和 GROUP BY 一起使用。

4.12.6 层次查询的限制

1. START WITH 子句中不能使用层次查询的所有伪列、层次查询函数、操作符；

2. ORDER SIBLINGS BY 子句中不能使用层次查询的所有伪列、层次查询函数、操作符、ROWNUM 以及子查询；

3. 层次查询子句不能使用伪列 CONNECT_BY_ISLEAF、CONNECT_BY_ISCYCLE，SYS_CONNECT_BY_PATH 伪函数和 CONNECT_BY_ROOT 操作符；

4. JOIN ON 子句中不允许出现层次查询的所有伪列、层次查询函数；

5. PRIOR、CONNECT_BY_ROOT 操作符后以及 SYS_CONNECT_BY_PATH 第一个参数不能使用层次查询的所有伪列、层次查询函数、操作符、ROWNUM 以及子查询。但 SYS_CONNECT_BY_PATH 的第一个参数允许出现 LEVEL 伪列且第二个参数必须是常量字符串，CONNECT_BY_ROOT LEVEL 也被允许；

6. 函数 SYS_CONNECT_BY_PATH 的最大返回长度为 8188，超长就会报错。函数

SYS_CONNECT_BY_PATH 在一个查询语句中最多使用个数为 64；

7. INI 参数 CNNTB_MAX_LEVEL 表示支持层次查询的最大层次，默认为 20000。该参数的有效取值为[1, 100000]。

例

对 OTHER.DEPARTMENT 数据进行层次查询，HIGH_DEP 表示上级部门；DEP_NAME 表示部门名称。

层次数据所建立起来的树形结构如下图：

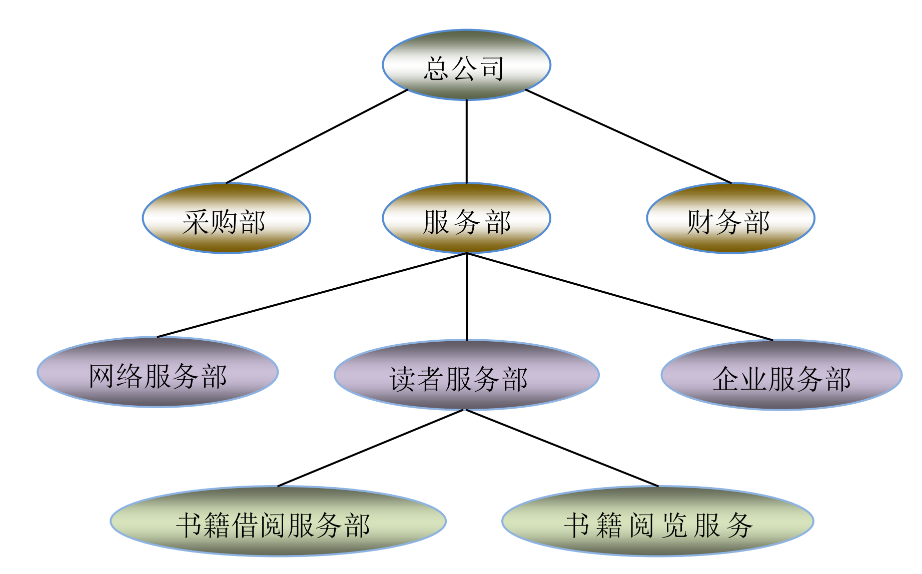


图 4.12.1 层次数据树形结构图

1. 不带起始选择根节点起始条件的层次查询

```
SELECT HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME = HIGH_DEP;
```

查询结果如表 4.12.1 所示。

表 4.12.1

序号	HIGH_DEP	DEP_NAME
1	NULL	总公司
2	总公司	服务部
3	服务部	网络服务部
4	服务部	读者服务部
5	读者服务部	书籍借阅服务部
6	读者服务部	书籍阅览服务部
7	服务部	企业服务部
8	总公司	采购部
9	总公司	财务部
10	总公司	财务部
11	总公司	采购部
12	总公司	服务部
13	服务部	网络服务部
14	服务部	读者服务部
15	读者服务部	书籍借阅服务部
16	读者服务部	书籍阅览服务部

17	服务部	企业服务部
18	服务部	企业服务部
19	服务部	读者服务部
20	读者服务部	书籍借阅服务部
21	读者服务部	书籍阅览服务部
22	服务部	网络服务部
23	读者服务部	书籍阅览服务部
24	读者服务部	书籍借阅服务部

结果是以表中所有的节点为根节点进行先根遍历进行层次查询。

2. 带起始选择根节点起始条件的层次查询

```
SELECT HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT CONNECT BY PRIOR
DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司';
```

查询结果如表 4.12.2 所示。

表 4.12.2

序号	HIGH_DEP	DEP_NAME
1	NULL	总公司
2	总公司	服务部
3	服务部	网络服务部
4	服务部	读者服务部
5	读者服务部	书籍借阅服务部
6	读者服务部	书籍阅览服务部
7	服务部	企业服务部
8	总公司	采购部
9	总公司	财务部

3. 层次查询伪列的使用

在层次查询中，伪列的使用可以更明确层次数据之间的关系。

```
SELECT LEVEL,
CONNECT_BY_ISLEAF ISLEAF,
CONNECT_BY_ISCYCLE ISCYCLE,
HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT
CONNECT BY PRIOR DEP_NAME=HIGH_DEP
START WITH DEP_NAME='总公司';
```

查询结果如表 4.12.3 所示。

表 4.12.3

序号	LEVEL	ISLEAF	ISCYCLE	HIGH_DEP	DEP_NAME
1	1	0	0	NULL	总公司
2	2	0	0	总公司	服务部
3	3	1	0	服务部	网络服务部
4	3	0	0	服务部	读者服务部
5	4	1	0	读者服务部	书籍借阅服务部
6	4	1	0	读者服务部	书籍阅览服务部

7	3	1	0	服务部	企业服务部
8	2	1	0	总公司	采购部
9	2	1	0	总公司	财务部

通过伪列，可以清楚地看到层次数据之间的层次结构。

4．含有过滤条件的层次查询

在层次查询中加入过滤条件，将会先进行层次查询，然后进行过滤。

```
SELECT LEVEL,* FROM OTHER.DEPARTMENT WHERE HIGH_DEP = '总公司' CONNECT BY PRIOR DEP_NAME=HIGH_DEP;
```

查询结果如表 4.12.4 所示。

表 4.12.4

序号	LEVEL	HIGH_DEP	DEP_NAME
1	2	总公司	服务部
2	2	总公司	采购部
3	2	总公司	财务部
4	1	总公司	财务部
5	1	总公司	采购部
6	1	总公司	服务部

5．含有排序子句的层次查询

在层次查询中加入排序，查询将会按照排序子句指明的要求排序，不再按照层次查询的排序顺序排序。

```
SELECT * FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司' ORDER BY HIGH_DEP;
```

查询结果如表 4.12.5 所示。

表 4.12.5

序号	HIGH_DEP	DEP_NAME
1	NULL	总公司
2	读者服务部	书籍阅览服务部
3	读者服务部	书籍借阅服务部
4	服务部	读者服务部
5	服务部	企业服务部
6	服务部	网络服务部
7	总公司	财务部
8	总公司	采购部
9	总公司	服务部

6．含层内排序子句的层次查询

在层次查询中加入 ORDER SIBLINGS BY，查询会对相同层次的数据进行排序后，深度优先探索返回数据，即 LEVEL 相同的数据进行排序。

```
SELECT HIGH_DEP, DEP_NAME, LEVEL FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司' ORDER SIBLINGS BY DEP_NAME;
```

查询结果如表 4.12.6 所示。

表 4.12.6

序号	HIGH_DEP	DEP_NAME	LEVEL
1	NULL	总公司	1
2	总公司	财务部	2
3	总公司	采购部	2
4	总公司	服务部	2
5	服务部	读者服务部	3
6	读者服务部	书籍借阅服务部	4
7	读者服务部	书籍阅览服务部	4
8	服务部	企业服务部	3
9	服务部	网络服务部	3

7. CONNECT_BY_ROOT 操作符的使用

CONNECT_BY_ROOT 操作符之后跟某列的列名，例如：

```
CONNECT_BY_ROOT DEP_NAME
```

进行如下查询：

```
SELECT CONNECT_BY_ROOT DEP_NAME,* FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司' ;
```

查询结果如表 4.12.7 所示。

表 4.12.7

序号	CONNECT_BY_ROOT DEP_NAME	HIGH_DEP	DEP_NAME
1	总公司	NULL	总公司
2	总公司	总公司	服务部
3	总公司	服务部	网络服务部
4	总公司	服务部	读者服务部
5	总公司	读者服务部	书籍借阅服务部
6	总公司	读者服务部	书籍阅览服务部
7	总公司	服务部	企业服务部
8	总公司	总公司	采购部
9	总公司	总公司	财务部

8. SYS_CONNECT_BY_PATH 函数的使用

函数的使用方式，如：

```
SYS_CONNECT_BY_PATH(DEP_NAME, '/')
```

进行如下查询：

```
SELECT SYS_CONNECT_BY_PATH(DEP_NAME, '/') PATH,* FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司' ;
```

查询结果如表 4.12.8 所示。

表 4.12.8

序号	PATH	HIGH_DEP	DEP_NAME
1	/总公司	NULL	总公司
2	/总公司/服务部	总公司	服务部
3	/总公司/服务部/网络服务部	服务部	网络服务部
4	/总公司/服务部/读者服务部	服务部	读者服务部

5	/总公司/服务部/读者服务部/书籍借阅服务部	读者服务部	书籍借阅服务部
6	/总公司/服务部/读者服务部/书籍阅览服务部	读者服务部	书籍阅览服务部
7	/总公司/服务部/企业服务部	服务部	企业服务部
8	/总公司/采购部	总公司	采购部
9	/总公司/财务部	总公司	财务部

4.13 并行查询

达梦支持并行查询技术。首先设置好如下三个 INI 参数，之后执行 SQL 语句，即可执行并行查询。三个 INI 参数解释如下表。

表 4.13.1 并行查询相关参数

参数名	缺省值	说明
MAX_PARALLEL_DEGREE	1	用来设置最大并行任务个数。取值范围：1~128。缺省值 1，表示无并行任务。全局有效。当 PARALLEL_POLICY 值为 1 时该参数值才有效。
PARALLEL_POLICY	0	用来设置并行策略。取值范围：0、1 和 2，缺省为 0。其中，0 表示不支持并行；1 表示自动配置并行工作线程个数（与物理 CPU 核数相同）；2 表示手动设置并行工作线程数。当 PARALLEL_POLICY 值为 2 时，需手动指定当前并行任务个数。
PARALLEL_THRD_NUM	10	用来设置并行工作线程个数。取值范围：1~1024。仅当 PARALLEL_POLICY 值为 2 时才启用此参数。

注：当处于 DMSQL 程序调试状态时，并行查询的相关设置均无效。

其中，并行任务数也可以在 SQL 语句中使用“PARALLEL”关键字特别指定。如果单条查询语句没有特别指定，则依然使用默认并行任务个数。“PARALLEL”关键字的用法为在数据查询语句的 SELECT 关键字后增加 HINT 子句。

语法格式

```
/*+ PARALLEL([<表名>] <并行任务个数>) */
```

使用说明

对于无特殊要求的并行查询用户，可以使用默认并行任务数 MAX_PARALLEL_DEGREE。只需要在 INI 参数中设置好对应参数，然后执行 SQL 查询语句，就可以启用并行查询。

举例说明

例 1 将 PARALLEL_POLICY 设置为 0，表示不支持并行查询。此时，另外两个参数不起任何作用。

```
PARALLEL_POLICY            0
```

例 2 将 PARALLEL_POLICY 设置为 1，表示自动配置并行工作线程个数，因此，只要设置下面 2 个参数就可以。

```
MAX_PARALLEL_DEGREE          3
PARALLEL_POLICY              1
```

然后，执行 SQL 语句。

```
SELECT * FROM SYSOBJECTS;         ---- 本条语句使用默认并行任务数 3
```

当然，如果单条查询语句不想使用默认并行任务数 3，可以通过在 SQL 语句中增加 HINT，通过“PARALLEL”关键字特别指定。本条语句使用特别指定的并行任务数 4，例如：

```
SELECT /*+ PARALLEL(4) */ * FROM SYSOBJECTS;
```

例 3 将 PARALLEL_POLICY 设置为 2，表示手动配置并行工作线程个数，因此，指定

如下 2 个参数。

```
PARALLEL_POLICY              2
PARALLEL_THRD_NUM             4
```

然后，在执行 SQL 语句时，需手动指定当前并行任务个数。若不指定，将不使用并行。

```
SELECT /*+ PARALLEL(2) */ * FROM SYSOBJECTS;    ----本条语句使用并行任务数 2。
```

4.14 ROWNUM

ROWNUM 是一个虚假的列，表示从表中查询的行号，或者连接查询的结果集行数。它将被分配为 1，2，3，4，...N，N 是行的数量。通过使用 ROWNUM 可以限制查询返回的行数。例如，以下语句执行只会返回前 5 行数据。

```
SELECT * FROM RESOURCES.EMPLOYEE WHERE ROWNUM < 6;
```

一个 ROWNUM 值不是被永久的分配给一行。表中的某一行并没有标号，不可以查询 ROWNUM 值为 5 的行。ROWNUM 值只有当被分配之后才会增长，并且初始值为 1。即只有满足一行后，ROWNUM 值才会加 1，否则只会维持原值不变。因此，以下语句在任何时候都不能返回数据。

```
SELECT * FROM RESOURCES.EMPLOYEE WHERE ROWNUM > 11;
SELECT * FROM RESOURCES.EMPLOYEE WHERE ROWNUM = 5;
```

ROWNUM 的一个重要作用是控制返回结果集的规模，可以避免查询在磁盘中排序。

因为 ROWNUM 值的分配是在查询的谓词解析之后，任何排序和聚合之前进行的。因此，在排序和聚合使用 ROWNUM 时需要注意，可能得到并非预期的结果，例如：

```
SELECT * FROM RESOURCES.EMPLOYEE WHERE ROWNUM < 11 ORDER BY EMPLOYEEID;
```

以上语句只会对 EMPLOYEE 表前 10 行数据按 EMPLOYEEID 排序输出，并不是表的所有数据按 EMPLOYEEID 排序后输出前 10 行，要实现后者，需要使用如下语句：

```
SELECT * FROM (SELECT * FROM RESOURCES.EMPLOYEE ORDER BY EMPLOYEEID) WHERE ROWNUM < 11;
SELECT TOP 10 * FROM RESOURCES.EMPLOYEE ORDER BY EMPLOYEEID;
```

使用说明

1. 在查询中，ROWNUM 可与任何数字类型表达式进行比较及运算，但不能出现在含 OR 的布尔表达式中，否则报错处理；
2. ROWNUM 可以在非相关子查询中使用；当参数 ENABLE_RQ_TO_INV 等于 1 时，部分相关子查询支持使用；
3. 在非相关子查询中，ROWNUM 只能实现与 TOP 相同的功能，因此子查询不能含 ORDER BY 和 GROUP BY；
4. ROWNUM 所处的子谓词只能为如下形式： ROWNUM op exp，exp 的类型只能是立即数、参数和变量值，op ∈ {<， <=， >， >=， =， <>}。

4.15 数组查询

在 DM 中，可以通过查询语句查询数组信息。即<FROM 子句>中<普通表>使用数组。语法如下：

```
FROM ARRAY <数组>
```

目前 DM 只支持一维数组的查询。

数组类型可以是记录类型和普通数据库类型。如果为记录类型的数组，则记录的成员都

必须为标量（基本）数据类型。记录类型数组查询出来的列名为记录类型每一个属性的名字。普通数据库类型查询出来的列名均为“COLUMN_VALUE”。

例 1 查看数组

```
SELECT * FROM ARRAY NEW INT[2]{1};
```

返回结果为:

```
COLUMN_VALUE
1
NULL
```

例 2 数组与表的连接

```
DECLARE
   TYPE rrr IS RECORD (x INT, y INT);
   TYPE ccc IS ARRAY rrr[];
   c ccc;
BEGIN
   c = NEW rrr[2];
   FOR i IN 1..2 LOOP
      c[i].x = i;
      c[i].y = i*2;
   END LOOP;
   SELECT arr.x, o.name  FROM ARRAY c arr, SYSOBJECTS o WHERE arr.x = o.id;
END;
```

返回结果为:

```
X              NAME
1              SYSINDEXES
2              SYSCOLUMNS
```

4.16 查看执行计划与执行跟踪统计

4.16.1 EXPLAIN

EXPLAIN 语句可以查看 DML 语句的执行计划。

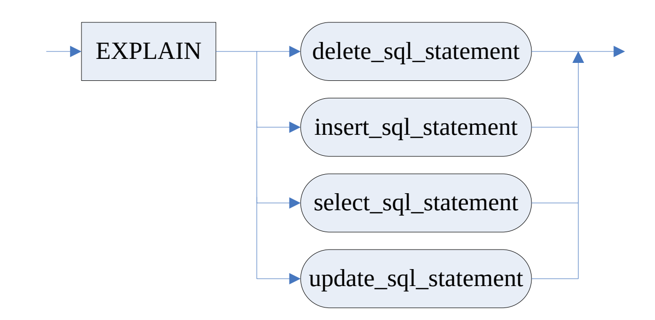
语法格式

```
EXPLAIN <SQL 语句>;
<SQL 语句> ::= <删除语句> | <插入语句> | <查询语句> | <更新语句>
```

参数

1. <删除语句>　指数据删除语句；
2. <插入语句>　指数据插入语句；
3. <查询语句>　指查询语句；
4. <更新语句>　指数据更新语句。

图例



语句功能

供用户查看执行计划。

举例说明

例 显示如下语句的查询计划：

```
EXPLAIN SELECT NAME,schid
FROM SYSOBJECTS
WHERE SUBTYPE$='STAB' AND NAME
 NOT IN (
SELECT NAME FROM SYSOBJECTS WHERE NAME IN (SELECT NAME FROM SYSOBJECTS WHERE SUBTYPE$='STAB') AND TYPE$='DSYNOM')
```

可以看到返回结果：

```
1   #NSET2: [2, 1, 142]
2     #PRJT2: [2, 1, 142]; exp_num(2), is_atom(FALSE)
3       #HASH LEFT SEMI JOIN2: [2, 1, 142];  (ANTI),
4         #SLCT2: [0, 20, 142]; SYSOBJECTS.SUBTYPE$ = STAB
5           #CSCN2: [0, 807, 142]; SYSINDEXSYSOBJECTS(SYSOBJECTS)
6         #PRJT2: [1, 1, 138]; exp_num(1), is_atom(FALSE)
7           #HASH LEFT SEMI JOIN2: [1, 1, 138];
8                         #CSEK2:   [0,   20,   138];   scan_type(ASC), SYSINDEXSYSOBJECTS(SYSOBJECTS), scan_range[(DSYNOM,min,min),(DSYNOM,max,max))
9             #SLCT2: [0, 20, 138]; SYSOBJECTS.SUBTYPE$ = STAB
10              #CSCN2: [0, 807, 138]; SYSINDEXSYSOBJECTS(SYSOBJECTS)
```

4.16.2 EXPLAIN FOR

EXPLAIN FOR 语句也用于查看 DML 语句的执行计划，不过执行计划以结果集的方式返回。

EXPLAIN FOR 显示的执行计划信息更加丰富，除了常规计划信息，还包括创建索引建议、分区表的起止分区信息等。重要的是，语句的计划保存在数据表中，方便用户随时查看，进行计划对比分析，可以作为性能分析的一种方法。

语法格式

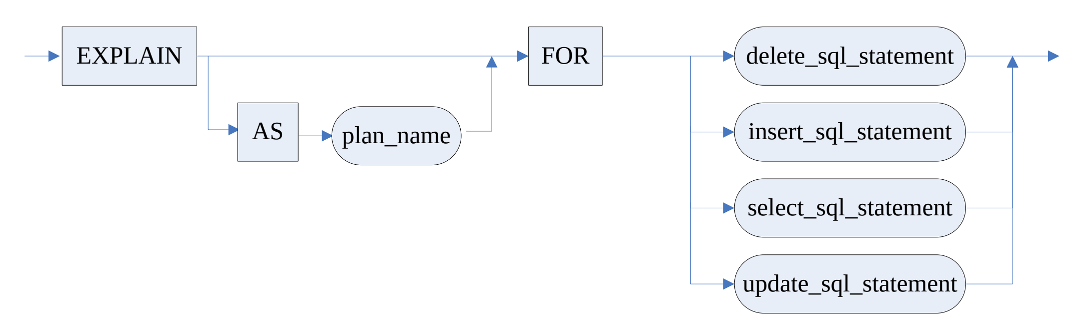
```
EXPLAIN  [AS 计划名称]  FOR <SQL 语句>;
<SQL 语句> ::= <删除语句> | <插入语句> | <查询语句> | <更新语句>
```

参数

1. <删除语句>　　指数据删除语句；

2. <插入语句>　　指数据插入语句；
3. <查询语句>　　指查询语句；
4. <更新语句>　　指数据更新语句。

图例



语句功能

供用户以结果集的方式查看执行计划。

举例说明

例 以结果集的方式显示如下语句的查询计划：

```
EXPLAIN FOR
SELECT NAME, SCHID FROM  SYS.SYSOBJECTS WHERE SUBTYPE$='STAB';
```

4.17 SAMPLE 子句

DM 通过 SAMPLE 子句实现数据采样功能。

语法格式

```
<SAMPLE 子句>::=SAMPLE (<表达式>) |
                SAMPLE (<表达式>) SEED (<表达式>) |
                SAMPLE BLOCK (<表达式>) |
                SAMPLE BLOCK (<表达式>) SEED (<表达式>)
```

参数

1. <表达式>输入整数与小数均可；

2. SAMPLE (<表达式>) 按行采样。<表达式>表示采样百分比，取值范围[0.000001,100)。重复执行相同语句，返回的结果不要求一致；

3. SAMPLE (<表达式>) SEED (<表达式>) 按行采样，并指定种子。其中 SEED (<表达式>)表示种子，取值范围[0, 4294967295]。重复执行相同的语句，每次返回相同的结果集；

4. SAMPLE BLOCK (<表达式>) 按块(页)采样。<表达式>表示采样百分比，取值范围[0.000001,100)。重复执行相同语句，返回的结果不要求一致，允许返回空集；

5. SAMPLE BLOCK (<表达式>) SEED (<表达式>) 按块(页)采样，并指定种子。其中，BLOCK (<表达式>)表示采样百分比，取值范围[0.000001,100)。SEED (<表达式>)表示种子，取值范围[0, 4294967295]。重复执行相同语句，每次返回相同的结果集。

使用说明

1. SAMPLE 只能出现在单表或仅包含单表的视图后面；
2. 包含过滤条件的 SAMPLE 查询，是对采样后的数据再进行过滤；
3. 不能对连接查询、子查询使用 SAMPLE 子句。

举例说明

例 对PERSON.ADDRESS表按行进行种子为5的10%采样。

```
SELECT * FROM PERSON.ADDRESS SAMPLE(10) SEED(5);
```

可以看到返回结果：

```
ADDRESSID      ADDRESS1       ADDRESS2      CITY         POSTALCODE
16          洪山区光谷软件园       C1_501   武汉市洪山区      430073
```

4.18 水平分区表查询

SELECT 语句从水平分区子表中检索数据，称水平分区子表查询，即<对象名>中使用的是<分区表>。水平分区父表的查询方式和普通表完全一样。

```
<分区表>::=
        [<模式名>.]<基表名> PARTITION (<一级分区名>) |
        [<模式名>.]<基表名> SUBPARTITION (<多级子分区名>)
```

参数

1. <基表名> 水平分区表父表名称；
2. <一级分区名> 水平分区表一级分区的名字；
3. <子分区名> 由水平分区表中多级分区名字逐级通过下划线"_"连接在一起的组合名称，例如 P1_P2_P3，其中 P1 是一级分区名、P2 是二级分区名、P3 是三级分区名。

使用说明

如果 HASH 分区不指定分区表名，而是通过指定哈希分区个数来建立哈希分区表，PARTITIONS 后的数字表示哈希分区的分区数，使用这种方式建立的哈希分区表分区名是匿名的，DM 统一使用 DMHASHPART+分区号（从 0 开始）作为分区名。

举例说明

例 1 查询一个 LIST-RANGE 三级水平分区表。

```
DROP TABLE STUDENT;
CREATE TABLE STUDENT(
NAME VARCHAR(20),
AGE INT,
SEX VARCHAR(10) CHECK (SEX IN ('MAIL','FEMAIL')),
GRADE INT CHECK (GRADE IN (7,8,9))
)
PARTITION BY LIST(GRADE)
  SUBPARTITION BY LIST(SEX) SUBPARTITION TEMPLATE
  (
     SUBPARTITION Q1 VALUES('MAIL'),
     SUBPARTITION Q2 VALUES('FEMAIL')
  ),
  SUBPARTITION BY RANGE(AGE) SUBPARTITION TEMPLATE
  (
     SUBPARTITION R1 VALUES LESS THAN (12),
     SUBPARTITION R2 VALUES LESS THAN (15),
     SUBPARTITION R3 VALUES LESS THAN (MAXVALUE)
   )
(
  PARTITION P1 VALUES (7),
```

```
  PARTITION P2 VALUES (8),
  PARTITION P3 VALUES (9)
);

SELECT * FROM STUDENT;                                    --查询水平分区父表
SELECT * FROM STUDENT PARTITION(P1);                      --查询一级分区子表
SELECT * FROM STUDENT SUBPARTITION(P1_Q1);                --查询二级分区子表
SELECT * FROM STUDENT SUBPARTITION(P1_Q1_R1);             --查询三级分区子表
```

例2 查询一个指定HASH分区名的水平分区表。

```
CREATE TABLESPACE TS1 DATAFILE 'TS1.DBF' SIZE 128;
CREATE TABLESPACE TS2 DATAFILE 'TS2.DBF' SIZE 128;
CREATE TABLESPACE TS3 DATAFILE 'TS3.DBF' SIZE 128;
CREATE TABLESPACE TS4 DATAFILE 'TS4.DBF' SIZE 128;

DROP TABLE CP_TABLE_HASH  CASCADE;
CREATE TABLE CP_TABLE_HASH(
            C1      INT,
            C2      VARCHAR(256),
            C3      DATETIME,
            C4      BLOB
            )
            PARTITION BY HASH (C1)
            SUBPARTITION BY HASH(C2)
            SUBPARTITION TEMPLATE
            (SUBPARTITION PAR1 STORAGE (ON MAIN),
             SUBPARTITION PAR2 STORAGE (ON TS1),
             SUBPARTITION PAR3 STORAGE (ON TS2),
             SUBPARTITION PAR4)
            (PARTITION PAR1 STORAGE (ON MAIN),
             PARTITION PAR2 STORAGE (ON TS1),
             PARTITION PAR3 STORAGE (ON TS2),
             PARTITION PAR4)
            STORAGE (ON TS4) ;

SELECT * FROM CP_TABLE_HASH PARTITION(PAR1);              --查询一级分区子表
SELECT * FROM CP_TABLE_HASH SUBPARTITION(PAR1_PAR1);      --查询二级分区子表
```

例 3 查询一个指定 HASH 分区数的水平分区，查询 CP_TABLE_HASH01 第一个分区的数据。

```
DROP TABLE CP_TABLE_HASH01  CASCADE;
CREATE TABLE CP_TABLE_HASH(
            C1      INT,
            C2      VARCHAR(256),
            C3      DATETIME,
            C4      BLOB
```

```
            )
            PARTITION BY HASH (C1)
            PARTITIONS 4 STORE IN (TS1, TS2, TS3, TS4);
SELECT * FROM CP_TABLE_HASH PARTITION (DMHASHPART0); --查询一级分区子表
```

第 5 章 数据的插入、删除和修改

DM_SQL 语言的数据更新语句包括：数据插入、数据修改和数据删除三种语句，其中数据插入和修改两种语句使用的格式要求比较严格。在使用时要求对相应基表的定义，如列的个数、各列的排列顺序、数据类型及关键约束、唯一性约束、引用约束、检查约束的内容均要了解得很清楚，否则就很容易出错。下面将分别对这三种语句进行讨论。在讨论中，如不特别说明，各例均使用示例库 BOOKSHOP，用户均为建表者 SYSDBA。

5.1 数据插入语句

数据插入语句用于向已定义好的表中插入单个或成批的数据。

INSERT 语句有两种形式。一种形式是值插入，即构造一行或者多行，并将它们插入到表中；另一种形式为查询插入，它通过<查询表达式>返回一个查询结果集以构造要插入表的一行或多行。

数据插入语句的语法格式如下：

语法格式

```
<插入表达式>::=
[@]INSERT  <single_insert_stmt> | <multi_insert_stmt>;

<single_insert_stmt>::=[INTO]  <full_tv_name>  [<t_alias>]  <insert_tail>
[<return_into_obj>]
<full_tv_name>::=
    | <单表引用>  [<LINK|@> <dblink_name>]
    | [<模式名>.]<基表名>  INDEX <索引名>
    | [<模式名>.]<基表名>  PARTITION (<分区名>)
| <子查询表达式>
<单表引用>::=[<模式名>.]<基表或视图名>
<基表或视图名>::=<基表名>|<视图名>
<子查询表达式>::=(<查询表达式>) [[AS] <表别名>]
<t_alias>::=[AS] <表别名>
<insert_tail>::= [(<列名>{,<列名>})]<insert_action>
<insert_action>::= VALUES <ins_value>
     | <查询表达式>|(<查询表达式>)
    | (<select_clause>)
    | DEFAULT VALUES
    | TABLE <full_tv_name>
<return_into_obj>::=
     <RETURN|RETURNING><expr{,expr}>INTO <data_item {,data_item }>
    |<RETURN|RETURNING><expr{,expr}>BULK    COLLECT    INTO    <data_item
{,data_item}>
<select_clause> 参见第四章
```

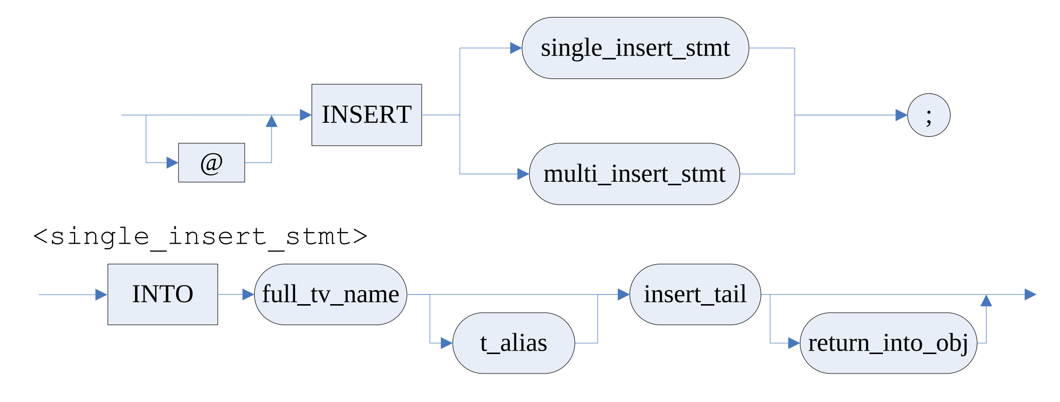
```
    <multi_insert_stmt>::=ALL <multi_insert_into_list> <查询表达式>
            |[ALL|FIRST]<multi_insert_into_condition_list>
[<multi_insert_into_else>]<查询表达式>
    <multi_insert_into_list>::= <insert_into_single>{<insert_into_single>}
    <insert_into_single>::=
        INTO <full_tv_name> [<t_alias>] [(<列名>{,<列名>})][VALUES <ins_value>]
    <ins_value>::=
     (<expr|DEFAULT>{,<expr|DEFAULT>}){,(<expr|DEFAULT>{,<expr| DEFAULT>})}
    <multi_insert_into_condition_list> ::=
    <insert_into_single_condition>{,< insert_into_single_condition>}
    <INSERT_INTO_SINGLE_CONDITION>::=
     WHEN <BOOL_EXP> THEN <MULTI_INSERT_INTO_LIST>
    <MULTI_INSERT_INTO_ELSE>::= ELSE <MULTI_INSERT_INTO_LIST>
```

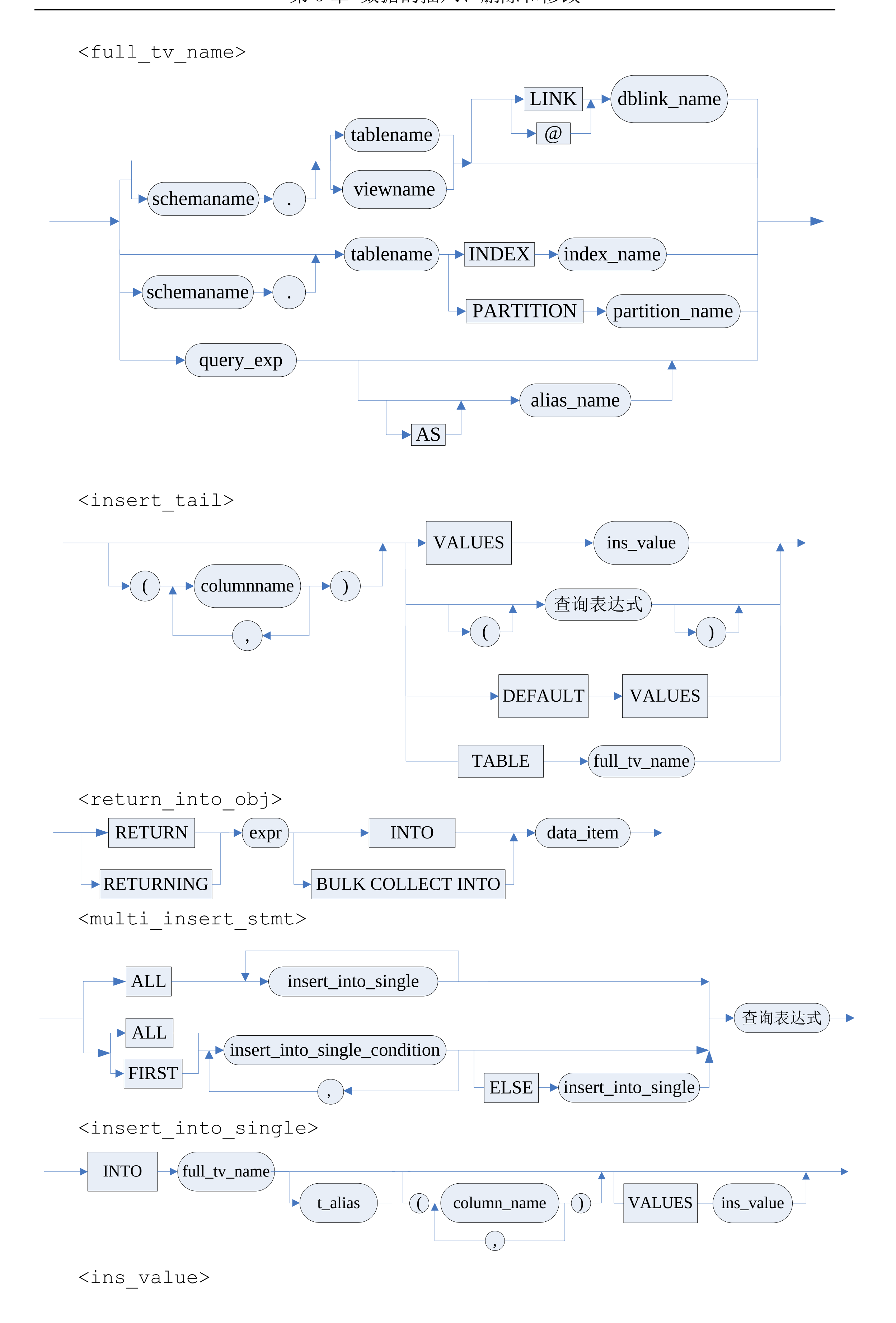
参数

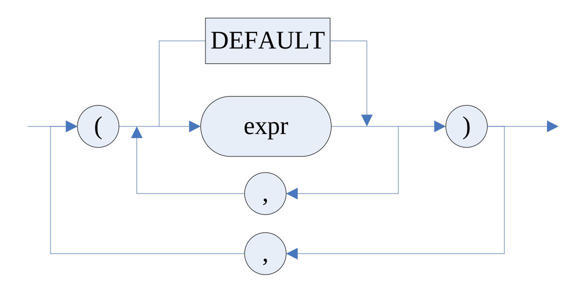
1. <模式名> 指明该表或视图所属的模式，缺省为当前模式；
2. <基表名> 指明被插入数据的基表的名称；
3. <视图名> 指明被插入数据的视图的名称，实际上 DM 将数据插入到视图引用的基表中；
4. <列名> 表或视图的列的名称。在插入的记录中，这个列表中的每一列都被 VALUES 子句或查询说明赋一个值。如果在此列表中省略了表的一个列名，则 DM 用先前定义好的缺省值插入到这一列中。如果此列表被省略，则在 VALUES 子句和查询中必须为表中的所有列指定值；
5. <INS_VALUE> 指明在列表中对应的列的插入的列值，如果列表被省略了，插入的列值按照基表中列的定义顺序排列。所有的插入值和系统内部相关存储信息一起构成了一条记录，一条记录的长度不能大于页面大小的一半；
6. <查询表达式> 将一个 SELECT 语句所返回的记录插入表或视图的基表中，子查询中选择的列表必须和 INSERT 语句中列名清单中的列具有相同的数量；带有<查询表达式>的插入方式，称查询插入。插入中使用的<查询表达式>也称为查询说明。
7. @ 当插入的是大数据数据文件时，启用@。同时对应的<插入值>格式为：@'path'。比如：@INSERT INTO T1 VALUES(@'e:\DSC_1663.jpg')。@用法只能在 DIsql 中使用，客户端工具不支持；
8. <dblink_name> 表示创建的 dblink 名字，如果添加了该选项，则表示插入远程实例的表。

图例

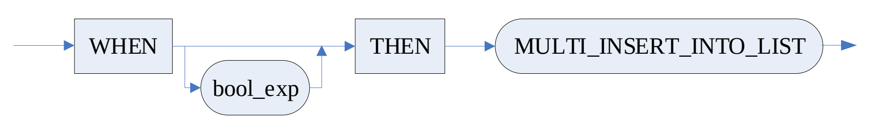
<插入表达式>




<full_tv_name>
LINK
dblink_name
@
tablename
viewname
schemaname
.
tablename
INDEX
index_name
schemaname
.
PARTITION
partition_name
query_exp
alias_name
AS
<insert_tail>
VALUES
ins_value
(
columnname
)
,
查询表达式
(
)
DEFAULT
VALUES
TABLE
full_tv_name
<return_into_obj>
RETURN
expr
INTO
data_item
RETURNING
BULK COLLECT INTO
<multi_insert_stmt>
ALL
insert_into_single
查询表达式
ALL
insert_into_single_condition
FIRST
ELSE
insert_into_single
,
<insert_into_single>
INTO
full_tv_name
t_alias
(
column_name
)
,
VALUES
ins_value
<ins_value>



<insert_into_single_condition>



使用说明

1. <基表名>或<视图名>后所跟的<列名>必须是该表中的列，且同一列名不允许出现两次，但排列顺序可以与定义时的顺序不一致；

2. <ins_value>中插入值的个数、类型和顺序要与<列名>一一对应；

3. 插入在指定值的时候，可以同时指定多行值，这种叫做多行插入或者批量插入。多行插入不支持列存储表；

4. 如果某一<列名>未在 INTO 子句后面出现，则新插入的行在这些列上将取空值或缺省值，如该列在基表定义时说明为 NOT NULL 时将会出错；

5. 如果<基表名>或<视图名>后没指定任何<列名>，则隐含指定该表或视图的所有列，这时，新插入的行必须在每个列上均有<插入值>；

6. 当使用<子查询表达式>作为 INSERT 的目标时，实际上是对查询表达式的基表进行操作，查询表达式的查询项必须都来源于同一个基表且不能是计算列，查询项所属的基表即是查询表达式的基表，如果查询表达式是带有连接的查询，那么对于连接中视图基表以外的表，连接列上必须是主键或者带有 UNIQUE 约束。不支持 PIVOT/UNPIVOT，不支持 UNION/UNION ALL 查询；

7. 如果两表之间存在引用和被引用关系时，应先插入被引用表的数据，再插入引用表的数据；

8. <查询表达式>是指用查询语句得到的一个结果集插入到插入语句中<表名>指定的表中，因此该格式的使用可供一次插入多个行，但插入时要求结果集的列与目标表要插入的列是一一对应的，不然会报错；

9. 多行插入时，对于存在行触发器的表，每一行都会触发相关的触发器；同样如果目标表具有约束，那么每一行都会进行相应的约束检查，只要有一行不满足约束，所有的值都不能插入成功；

10. 在嵌入方式下工作时，<ins_value>插入的值可以为主变量；

11. 如果插入对象是视图，同时在这个视图上建立了 INSTEAD OF 触发器，则会将插入操作转换为触发器所定义的操作；如果没有触发器，则需要判断这个视图是否可更新，如果不可更新则报错，否则是可以插入成功的；

12. RETURN INTO 返回列支持返回 ROWID；

13. RETURN INTO 语句中返回结果对象支持变量和数组。如果返回列为记录数组，则返回结果数只能为 1，且记录数组属性类型与个数须与返回列一致；如果为变量，则变量类型与个数与返回列一致；如果返回普通数组，则数组个数和数组元素类型与返回列一致；返回结果不支持变量、普通数组和记录数组混和使用；

14. 增删改语句当前修改表称为变异表（MUTATE TABLE），其调用函数中，不能对此

变异表进行插入操作；

15. BULK COLLECT 的作用是将检索结果批量地、一次性地赋给集合变量。与每次获取一条数据，并每次都要将结果赋值给变量相比，可以很大程度上的节省开销。使用 BULK COLLECT 时，INTO 后的变量必须是集合类型的。

举例说明

例 在 VENDOR 表中插入一条供应商信息：账户号为 00，名称为华中科技大学出版社，活动标志为 1，URL 为空，信誉为 2。

```
INSERT  INTO  PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES ('00', '华中科技大学出版社', 1, '', 2);
```

如果需要同时多行插入，则可以用如下的 SQL 语句实现：

```
INSERT  INTO  PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES ('00', '华中科技大学出版社', 1, '', 2), ('00', '清华大学出版社', 1, '', 3);
```

在定义 VENDOR 表时，设定了检查约束：CHECK(CREDIT IN(1,2,3,4,5))，说明 CREDIT 只能是 1，2，3，4，5。在插入新数据或修改供应商的 CREDIT 时，系统按检查约束进行检查，如果不满足条件，系统将会报错，多行插入中，每一行都会做检查。

由于 DM 支持标量子查询，标量子查询允许用在标量值合法的地方，因此在数据插入语句的<插入值>位置允许出现标量子查询。

例 将书名为长征的图书的出版社插入到 VENDOR 表中。

```
INSERT  INTO  PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00',
(SELECT  PUBLISHER  FROM  PRODUCTION.PRODUCT  WHERE  NAME='长征'),1,'',1);
```

若是需要插入一批数据时，可使用带<查询说明>的插入语句，如下例所示。

例 构造一个新的基表，表名为 PRODUCT_SELL，用来显示出售的商品名称和购买用户名称，并将查询的数据插入此表中。

```
CREATE  TABLE  PRODUCTION.PRODUCT_SELL
( PRODUCTNAME  VARCHAR(50)  NOT  NULL,
  CUSTOMERNAME  VARCHAR(50)  NOT  NULL);
INSERT  INTO  PRODUCTION.PRODUCT_SELL
SELECT  DISTINCT  T1.NAME ,  T5.NAME
FROM  PRODUCTION.PRODUCT  T1,  SALES.SALESORDER_DETAIL  T2,
       SALES.SALESORDER_HEADER  T3,  SALES.CUSTOMER  T4,
       PERSON.PERSON  T5
WHERE  T1.PRODUCTID = T2.PRODUCTID  AND  T2.SALESORDERID = T3.SALESORDERID
        AND  T3.CUSTOMERID = T4.CUSTOMERID  AND  T4.PERSONID = T5.PERSONID;
```

该插入语句将已销售的商品名称和购买该商品的用户名称插入到新建的 PRODUCT_SELL 表中。查询结果如下表 5.1.1 所示。

表 5.1.1

PRODUCTNAME	CUSTOMERNAME
红楼梦	刘青
老人与海	刘青

值得一提的是，BIT 数据类型值的插入与其他数据类型值的插入略有不同：其他数据类型皆为插入的是什么就是什么，而 BIT 类型取值只能为 1/0，又同时能与整数、精确数值类型、不精确数值类型和字符串类型相容(可以使用这些数据类型对 BIT 类型进行赋值和比较)，取值时有一定的规则。

数值类型常量向 BIT 类型插入的规则是：非 0 数值转换为 1，数值 0 转换为 0。例如：

```
CREATE TABLE T10 (C BIT);
INSERT INTO T10 VALUES(1);      --插入 1
INSERT INTO T10 VALUES(0);      --插入 0
INSERT INTO T10 VALUES(1.2);    --插入 1
```

字符串类型常量向 BIT 类型插入的规则是：

全部由 0 组成的字符串转换为 0，其他全数字字符串(例如 123)，转换为 1。非全数字字符串(例如：1e1, 2a5, 3.14)也转换为 1。

```
INSERT INTO T10 VALUES(000);    --插入 0
INSERT INTO T10 VALUES(0);       --插入 0
INSERT INTO T10 VALUES(10);     --插入 1
INSERT INTO T10 VALUES(1.0);    --插入 1
```

5.2 数据修改语句

数据修改语句用于修改表中已存在的数据。

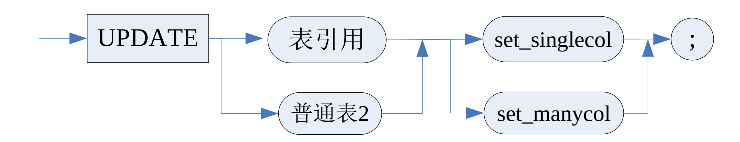
语法格式

```
UPDATE <单表引用>|<子查询> {<单列修改子句>|<多列修改子句>}
<子查询>::=[<普通表 2>]
<普通表 2>::= (<查询表达式>) [[AS] <表别名> [<新生列>]]
<单列修改子句>::= SET<列名>=<<值表达式>|DEFAULT>{,<列名>=<<值表达式>|DEFAULT>}[FROM <表引用>{,<表引用>}][WHERE <条件表达式>][<RETURN_INTO_OBJ>];
<多列修改子句>::= SET <列名>{,<列名>}= <SUBQUERY>;
<单表引用>::= [<模式名>.]<基表或视图名>
<基表或视图名>::= <基表名>|<视图名>
<结果对象>::= <数组>|<变量>
<RETURN_INTO_OBJ>::=
     <RETURN|RETURNING><列名>{,<列名>}INTO <结果对象>
    |<RETURN|RETURNING><列名>{,<列名>}BULK COLLECT INTO <结果对象>
```

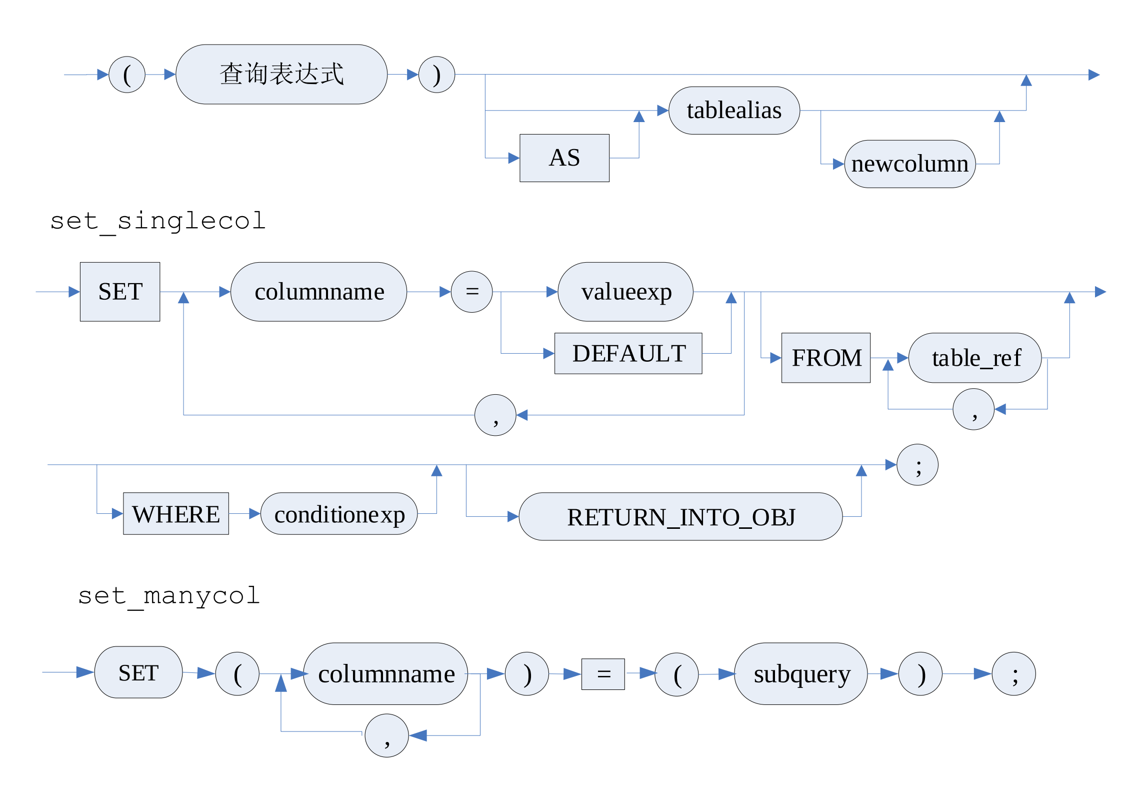
参数

1. <模式名>　　指明该表或视图所属的模式，缺省为当前用户的缺省模式；
2. <基表名>　　指明被修改数据的基表的名称；
3. <视图名>　　指明被修改数据的视图的名称，实际上 DM 对视图的基表更新数据；
4. <列名>　　表或视图中被更新列的名称，如果 SET 子句中省略列的名称，列的值保持不变；
5. <值表达式>　指明赋予相应列的新值；
6. <条件表达式>指明限制被更新的行必须符合指定的条件，如果省略此子句，则修改表或视图中所有的行。

图例



普通表 2



RETURN_INTO_OBJ：请参考第 4 章 数据查询语句

使用说明

1. SET 后的<列名>不能重复出现；

2. WHERE 子句也可以包含子查询。如果省略了 WHERE 子句，则表示要修改所有的元组；

3. 如果<列名>为被引用列，只有被引用列中未被引用列引用的数据才能被修改；如果<列名>为引用列，引用列的数据被修改后也必须满足引用完整性。在 DM 系统中，以上引用完整性由系统自动检查；

4. 执行基表的 UPDATE 语句触发任何与之相联系的 UPDATE 触发器；

5. 对于未指定 ENABLE ROW MOVEMENT 属性水平分区表的更新，如果更新后的值将导致记录所属分区发生修改，则不能进行更新；更新包含大字段的水平分区表时不允许分区更改。

6. 如果视图的定义查询中含有以下结构则不能更新视图：

   1) 联结运算；
   2) 集合运算符；
   3) GROUP BY 子句；
   4) 集函数；
   5) INTO 子句；
   6) 分析函数；
   7) HAVING 字句；
   8) 层次查询子句。

7. 如果更新对象是视图，同时在这个视图上建立了 INSTEAD OF 触发器，则会将更新操作转换为触发器所定义的操作；如果没有触发器，则需要判断这个视图是否可更新，如果不可更新则报错，否则可以继续更新，如果上面的条件都满足，则可以更新成功；

8. RETURN INTO 不支持返回 ROWID 列；

9. RETURN INTO 语句中返回列如果是更新列，则返回值为列的新值。返回结果对象

支持变量和数组。如果返回列为记录数组，则返回结果数只能为 1.且记录数组属性类型和个数须与返回列一致；如果为变量，则变量类型与个数与返回列一致；如果返回普通数组，则数组个数与数组元素类型与返回列一致；返回结果不支持变量、普通数组和记录数组混和使用；

10．UPDATE 语句支持一次进行多列修改，多列修改存在以下限制：

1） 集合操作情况（UNION 等）：只有当查询语句为非相关子查询才支持集合操作；
2） 多列修改不支持 EXPLAIN 操作；
3） 子查询的结果不能多于 1 行数据。

11．如果更新为子查询，则存在以下限制：

1） 更新子查询对应的最终更新对象目前仅仅必须为基表；
2） 更新的子查询的查询结果必须保证所更新基表的唯一性特性，类似于更新视图是否可更新概念；
3） 更新子查询不支持多列更新；
4） 集合操作、DISTINCT 操作、集函数操作、带有 GROUP BY、CONNECT BY 等语句都不满足视图的更新性要求，报错；
5） 分区表暂不支持。

12．增删改语句当前修改表称为变异表（MUTATE TABLE），其调用函数中，不能对此变异表进行删除操作。

举例说明

例 将出版社为中华书局的图书的现在销售价格增加 1 元。

```
UPDATE  PRODUCTION.PRODUCT  SET  NOWPRICE = NOWPRICE + 1.0000
WHERE  PUBLISHER = '中华书局';
```

例 由于标量子查询允许用在标量值合法的地方，因此在数据修改语句的<值表达式>位置也允许出现标量子查询。下例将折扣高于 7.0 且出版社不是中华书局的图书的折扣设成出版社为中华书局的图书的平均折扣。

```
UPDATE  PRODUCTION.PRODUCT  SET  DISCOUNT =
( SELECT  AVG(DISCOUNT)
    FROM  PRODUCTION.PRODUCT
WHERE PUBLISHER = '中华书局')
WHERE  DISCOUNT > 7.0  AND  PUBLISHER != '中华书局';
```

注：自增列的修改例外，它一经插入，只要该列存储于数据库中，其值为该列的标识，不允许修改。关于自增列修改的具体情况，请参见 5.6 节—自增列的使用。

例 带 RETURN INTO 的更新语句。

```
CREATE TABLE T1(C1 INT,C2 INT,C3 INT);
DECLARE
  TYPE RRR IS RECORD(X INT, Y INT);
  TYPE CCC IS ARRAY RRR[];
  A INT;
  C CCC;
BEGIN
  C = NEW RRR[2];
  UPDATE T1 SET C2=4 WHERE C3 = 2 RETURN C1 INTO A;
  PRINT A;
  UPDATE T1 SET C2=5 WHERE C3 = 2 RETURN C1,C2 INTO C;
  SELECT * FROM ARRAY C;
```

```
    END;
```

例 使用一次进行多列修改的更新语句。

```
    UPDATE      PURCHASING.PURCHASEORDER_HEADER      SET(TAX,FREIGHT)=(SELECT
ORIGINALPRICE, NOWPRICE FROM PRODUCTION.PRODUCT WHERE NAME='长征');
```

5.3 数据删除语句

数据删除语句用于删除表中已存在的数据。

语法格式

```
DELETE  [FROM]  <表引用>
[WHERE <条件表达式>][RETURN <列名>{,<列名>} INTO <结果对象>,{<结果对象>}];
<表引用>::= [<模式名>.]<基表或视图名> |
            <子查询表达式>
<基表或视图名>::= <基表名>|<视图名>
<子查询表达式>::=(<查询表达式>) [[AS] <表别名> [<新生列>]]
<结果对象>::=<数组>|<变量>
```

参数

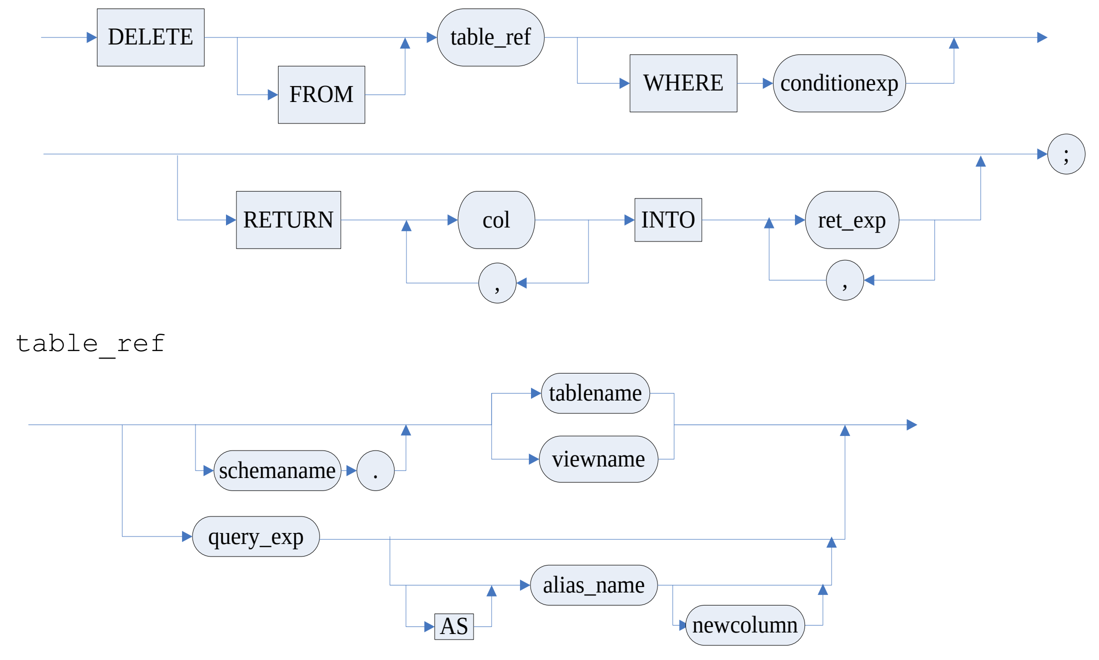
1. <模式名>　　指明该表或视图所属的模式，缺省为当前模式；

2. <基表名>　　指明被删除数据的基表的名称；

3. <视图名>　　指明被删除数据的视图的名称，实际上 DM 将从视图的基表中删除数据；

4. <条件表达式> 指明基表或视图的基表中被删除的记录须满足的条件。

图例



使用说明

1． 如果不带 WHERE 子句，表示删除表中全部元组，但表的定义仍在字典中。因此，DELETE 语句删除的是表中的数据，并未删除表结构；

2． 由于 DELETE 语句一次只能对一个表进行删除，因此当两个表存在引用与被引用关系时，要先删除引用表里的记录，只有引用表中无记录时，才能删被引用表中的记录，否

则系统会报错；

3． 执行与表相关的 DELETE 语句将触发所有定义在表上的 DELETE 触发器；

4． 如果视图的定义查询中包含以下结构之一，就不能从视图中删除记录：

1） 联结运算；

2） 集合运算符；

3） GROUP BY 子句；

4） 集函数；

5） INTO 语句；

6） 分析函数；

7） HAVING 语句；

8） CONNECT BY 语句。

5． 当<子查询表达式>作为 DELETE 的目标时，实际上是对查询表达式的基表进行操作，查询表达式的查询项必须都来源于同一个基表且不能是计算列，查询项所属的基表即是查询表达式的基表，如果查询表达式是带有连接的查询，那么对于连接中视图基表以外的表，连接列上必须是主键或者带有 UNIQUE 约束。不支持 PIVOT/UNPIVOT，不支持 UNION/UNION ALL 查询；

6． RETURN INTO 不支持返回 ROWID 列；

7． RETURN INTO 返回结果对象支持变量和数组。如果返回列为记录数组，则返回结果数只能为 1.且记录数组属性类型和个数须与返回列一致；如果为变量，则变量类型与个数与返回列一致；如果返回普通数组，则数组个数与数组元素类型与返回列一致；返回结果不支持变量、普通数组和记录数组混和使用；

8． 增删改语句当前修改表称为变异表（MUTATE TABLE），其调用函数中，不能对此变异表进行删除操作。

例 将没有分配部门的员工的住址信息删除。

```
DELETE FROM RESOURCES.EMPLOYEE_ADDRESS
WHERE EMPLOYEEID IN
( SELECT EMPLOYEEID
FROM RESOURCES.EMPLOYEE
WHERE EMPLOYEEID NOT IN
( SELECT EMPLOYEEID FROM RESOURCES.EMPLOYEE_DEPARTMENT));
```

5.4 MERGE INTO 语句

使用 MERGE INTO 语法可合并 UPDATE 和 INSERT 语句。通过 MERGE 语句，根据一张表（或视图）的连接条件对另外一张表（或视图）进行查询，连接条件匹配上的进行 UPDATE（可能含有 DELETE），无法匹配的执行 INSERT。其中，数据表包括：普通表、分区表、加密表、压缩表和堆表。

语法格式

```
MERGE INTO <MERGE_INTO_OBJ> [<表别名>] USING <表引用>ON (<条件判断表达式>)
<[<MERGE_UPDATE_CLAUSE >] [< MERGE_INSERT_CLAUSE>] >
<MERGE_INTO_OBJ> ::= <单表引用> | <子查询>
<单表引用> ::= [<模式名>.]<基表或视图名>
<子查询> ::= (<查询表达式>)
< MERGE_UPDATE_CLAUSE >::=WHEN MATCHED THEN UPDATE SET <SET_VALUE_LIST>
```

```
<WHERE_CLAUSE_NULL > [DELETE <WHERE_CLAUSE_NULL >]
    <MERGE_INSERT_CLAUSE>::=WHEN NOT MATCHED THEN INSERT [<FULL_COLUMN_LIST>]
VALUES <INS_VALUE_LIST > <WHERE_CLAUSE_NULL >;
    <表引用>::=<普通表> | <连接表>  详见《DM8_SQL 语言使用手册》第四章 数据查询语句
    <SET_VALUE_LIST> ::= <列名>=<值表达式| DEFAULT> {,<列名>=<值表达式| DEFAULT>}
    <WHERE_CLAUSE_NULL > ::= [WHERE <条件表达式>]
    <FULL_COLUMN_LIST>::= (<列名>{,<列名>})
    <INS_VALUE_LIST >::= (<插入值>{,<插入值>})
```

参数

1. <模式名> 指明该表或视图所属的模式，缺省为当前用户的缺省模式；
2. <基表名> 指明被修改数据的基表的名称；
3. <视图名> 指明被修改数据的视图的名称，实际上 DM 对视图的基表更新数据；
4. <查询表达式> 指明被修改数据的子查询表达式，不支持带有计算列、连接、pIVOT、UNPIVOT/UNION/UNION ALL 的查询，DM 实际是对子查询的基表进行数据更新；
5. <条件表达式> 指明限制被操作执行的行必须符合指定的条件，如果省略此子句，则对表或视图中所有的行进行操作。

图例

MERGE INTO [schemaname .] tablename | viewname | (query_exp) [t_ailas] USING

表引用 ON (conditionexp)

[merge_update_clause] [merge_insert_clause]

merge_update_clause

WHEN MATCHED THEN

UPDATE SET set_value_list [WHERE conditionexp]

[DELETE [WHERE conditionexp]]

merge_insert_clause

WHEN NOT MATCHED THEN INSERT

[full_column_list] VALUES ins_value_list [WHERE conditionexp]

使用说明

1．INTO 后为目标表，表示待更新、插入的表、可更新视图及可更新查询表达式；

2．USING 后为源表（普通表或可更新视图），表示用于和目标表匹配、更新或插入的数据源；

3．ON (<条件判断表达式>)表示目标表和源表的连接条件，如果目标表有匹配连接条件的记录则执行更新该记录，如果没有匹配到则执行插入源表数据；

4．MERGE_UPDATE_CLAUSE：当目标表和源表的 JOIN 条件为 TRUE 时，执行该语句；

1) 如果更新执行，更新语句会触发所有目标表上的 UPDATE 触发器，也会进行约束检查；
2) 可以指定更新条件，如果不符合条件就不会执行更新操作。更新条件既可以和源表相关，也可以和目标表相关，或者都相关；
3) DELETE 子句只删除目标表和源表的 JOIN 条件为 TRUE，并且是更新后的符合删除条件的记录，DELETE 子句不影响 INSERT 项插入的行。删除条件作用在更新后的记录上，既可以和源表相关，也可以和目标表相关，或者都相关。如果 JOIN 条件为 TRUE，但是不符合更新条件，并没有更新数据，那么 DELETE 将不会删除任何数据。当执行了删除操作，会触发目标表上的 DELETE 触发器，也会进行约束检查。

5．MERGE_INSERT_CLAUSE：当目标表和源表的 JOIN 条件为 FALSE 时，执行该语句。同时会触发目标表上的 INSERT 触发器，也会进行约束检查。可指定插入条件，插入条件只能在源表上设置；

6．MERGE_UPDATE_CLAUSE 和 MERGE_INSERT_CLAUSE 既可以同时指定，也可以只出现其中任何一个；

7．需要有对源表的 SELECT 权限，对目标表的 UPDATE/INSERT 权限，如果 UPDATE 子句有 DELETE，还需要有 DELETE 权限；

8．UPDATE 子句不能更新在 ON 连接条件中出现的列；

9．如果匹配到，源表中的匹配行必须唯一，否则报错；

10．<ins_value_list >不能包含目标表列；

11．插入的 WHERE 条件只能包含源表列。

举例说明

下面的例子把 T1 表中 C1 值为 2 的记录行中的 C2 列，更新为表 T2 中 C3 值为 2 的记录中 C4 列的值，同时把 T2 中 C3 列为 4 的记录行插入到 T1 中。

```
DROP TABLE T1;
DROP TABLE T2;
CREATE TABLE T1 (C1 INT,  C2 VARCHAR(20));
CREATE TABLE T2 (C3 INT,  C4 VARCHAR(20));
INSERT INTO T1 VALUES(1,'T1_1');
INSERT INTO T1 VALUES(2,'T1_2');
INSERT INTO T1 VALUES(3,'T1_3');
INSERT INTO T2 VALUES(2,'T2_2');
INSERT INTO T2 VALUES(4,'T2_4');
COMMIT;

MERGE INTO T1 USING T2 ON (T1.C1=T2.C3)
```

```
WHEN MATCHED THEN UPDATE SET T1.C2=T2.C4
WHEN NOT MATCHED THEN INSERT (C1,C2) VALUES(T2.C3, T2.C4);
```

下面的例子把 T1 表中 C1 值为 2，4 的记录行中的 C2 列更新为表 T2 中 C3 值为 2，4 的记录中 C4 列的值，同时把 T2 中 C3 列为 5 的记录行插入到了 T1 中。由于 UPDATE 带了 DELETE 子句，且 T1 中 C1 列值为 2 和 4 的记录行被更新过，而 C1 为 4 的行符合删除条件，最终该行会被删除掉。

```
DROP TABLE T1;
DROP TABLE T2;
CREATE TABLE T1 (C1 INT, C2 VARCHAR(20));
CREATE TABLE T2 (C3 INT, C4 VARCHAR(20));
INSERT INTO T1 VALUES(1,'T1_1');
INSERT INTO T1 VALUES(2,'T1_2');
INSERT INTO T1 VALUES(3,'T1_3');
INSERT INTO T1 VALUES(4,'T1_4');
INSERT INTO T2 VALUES(2,'T2_2');
INSERT INTO T2 VALUES(4,'T2_4');
INSERT INTO T2 VALUES(5,'T2_5');
COMMIT;

MERGE INTO T1 USING T2 ON (T1.C1=T2.C3)
WHEN MATCHED THEN UPDATE SET T1.C2=T2.C4 WHERE T1.C1 >= 2 DELETE WHERE T1.C1=4
WHEN NOT MATCHED THEN INSERT (C1,C2) VALUES(T2.C3, T2.C4);
```

5.5 伪列的使用

除了 4.12.2 层次查询相关伪列和 4.14 ROWNUM 中介绍的伪列外，DM 中还提供包括 ROWID、UID、USER、TRXID 等伪列。

5.5.1 ROWID

伪列从语法上和表中的列很相似，查询时能够返回一个值，但实际上在表中并不存在。用户可以对伪列进行查询，但不能插入、更新和删除它们的值。DM 支持的伪列有：ROWID，USER，UID，TRXID、ROWNUM 等。

DM 中行标识符 ROWID 用来标识数据库基表中每一条记录的唯一键值，标识了数据记录的确切的存储位置。如果用户在选择数据的同时从基表中选取 ROWID，在后续的更新语句中，就可以使用 ROWID 来提高性能。如果在查询时加上 FOR UPDATE 语句，该数据行就会被锁住，以防其他用户修改数据，保证查询和更新之间的一致性。例如：

```
SELECT  ROWID, VENDORID, NAME, CREDIT
FROM  PURCHASING.VENDOR
WHERE  NAME = '广州出版社';
--假设查询的 ROWID=CF06000000
UPDATE  PURCHASING.VENDOR  SET  CREDIT  = 2
WHERE  ROWID = 0XCF06000000;
```

5.5.2 UID 和 USER

伪列 USER 和 UID 分别用来表示当前用户的用户名和用户标识。

5.5.3 TRXID

伪列 TRXID 用来表示当前事务的事务标识。

5.5.4 SESSID

伪列 SESSID 用来表示当前会话的 ID 标识。

5.5.5 PHYROWID

伪列 PHYROWID 用来表示当前记录的物理存储信息。

PHYROWID 值由聚集 B 树或二级 B 树中物理记录的文件号、页号、页内槽号组成，能体现聚集 B 树或二级 B 树的存储信息，聚集 B 树记录的最高位为 1。

当查询语句中实际使用 CSCN、CSEK、BLKUP 操作符时，PHYROWID 内容是聚集 B 树中记录的物理存储地址；当查询语句中实际仅使用 SSEK、SSCN 操作符时，PHYROWID 内容是二级 B 树中记录的物理存储地址。

5.6 DM 自增列的使用

5.6.1 DM 自增列定义

1. 自增列功能定义

在表中创建一个自增列。该属性与 CREATE TABLE 语句一起使用，一个表只能有一个自增列。

语法格式

```
IDENTITY [ (种子, 增量) ]
```

参数

1. 种子　装载到表中的第一个行所使用的值；

2. 增量　增量值，该值被添加到前一个已装载的行的标识值上。增量值可以为正数或负数，但不能为 0。

使用说明

1. IDENTITY 适用于 INT(-2147483648～+2147483647)、BIGINT(-2^{63}～$+2^{63}-2$)类型的列。每个表只能创建一个自增列；

2. 不能对自增列使用 DEFAULT 约束；

3. 必须同时指定种子和增量值，或者二者都不指定。如果二者都未指定，则取默认值

(1,1)。若种子或增量为小数类型，报错；

4. 最大值和最小值为该列的数据类型的边界；

5. 建表种子和增量大于最大值或者种子和增量小于最小值时报错；

6. 自增列一旦生成，无法更新，不允许用 UPDATE 语句进行修改；

7. 临时表、列存储表、水平分区表不支持使用自增列。

2. 自增列查询函数

1) IDENT_SEED(函数)

语法格式

```
IDENT_SEED ('tablename')
```

功能：返回种子值，该值是在带有自增列的表中创建自增列时指定的。

参数：tablename 是带有引号的字符串常量，也可以是变量、函数或列名。tablename 的数据类型为 CHAR 或 VARCHAR。其含义是表名，可带模式名前缀。

返回类型：返回数据类型为 INT/NULL

2) IDENT_INCR(函数)

语法格式

```
IDENT_INCR ('tablename')
```

功能：返回增量值，该值是在带有自增列的表中创建自增列时指定的。

参数：tablename 是带有引号的字符串常量，也可以是变量、函数或列名。tablename 的数据类型为 CHAR 或 VARCHAR。其含义是表名，可带模式名前缀。

返回类型：返回数据类型为 INT/NULL

例 用自增列查询函数获得表 PERSON_TYPE 的自增列的种子和增量信息。

```
SELECT  IDENT_SEED('PERSON.PERSON_TYPE');
查询结果为：1
SELECT IDENT_INCR('PERSON.PERSON_TYPE');
查询结果为：1
```

5.6.2 SET IDENTITY_INSERT 属性

设置是否允许将显式值插入表的自增列中。

语法格式

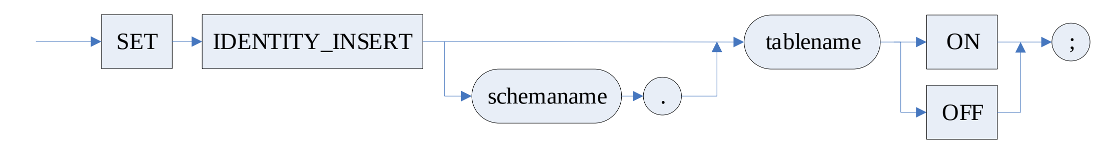
```
SET IDENTITY_INSERT [<模式名>.]<表名> ON | OFF;
```

参数

1. <模式名>　　指明表所属的模式，缺省为当前模式；

2. <表名>　　指明含有自增列的表名。

图例



使用说明

1. IDENTITY_INSERT 属性的默认值为 OFF。SET IDENTITY_INSERT 的设置是在执行或运行时进行的。当一个连接结束，IDENTITY_INSERT 属性将被自动还原为 OFF；

2. DM 要求一个会话连接中只有一个表的 IDENTITY_INSERT 属性可以设置为 ON，当设置一个新的表 IDENTITY_INSERT 属性设置为 ON 时，之前已经设置为 ON 的表会自动还原为 OFF。当一个表的 IDENTITY_INSERT 属性被设置为 ON 时，该表中的自动增量列的值由用户指定。如果插入值大于表的当前标识值(自增列当前值)，则 DM 自动将新插入值作为当前标识值使用，即改变该表的自增列当前值；否则，将不影响该自增列当前值；

3. 当设置一个表的 IDENTITY_INSERT 属性为 OFF 时，新插入行中自增列的当前值由系统自动生成，用户将无法指定；

4. 自增列一经插入，无法修改；

5. 手动插入自增列，除了将 IDENTITY_INSERT 设置为 ON，还要求在插入列表中明确指定待插入的自增列列名。插入方式与非 IDENTITY 表是完全一样的。如果插入时，既不指定自增列名也不给自增列赋值，则新插入行中自增列的当前值由系统自动生成。

举例说明

例 SET IDENTITY_INSERT 的使用。

1) PERSON_TYPE 表中的 PERSON_TYPEID 列是自增列，目前拥有的数据如表 5.6.1 所示。

表 5.6.1

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表

2) 在该表中插入数据，自增列的值由系统自动生成。

```
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('销售总监');
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('人力资源部经理');
```

插入结果如表 5.6.2 所示：

表 5.6.2

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	销售总监
6	人力资源部经理

3) 当插入数据并且要指定自增列的值时，必须要通过语句将 IDENTITY_INSERT 设置为 ON 时，插入语句中必须指定 PERSON_TYPEID 中要插入的列。例如：

```
SET IDENTITY_INSERT PERSON.PERSON_TYPE ON;
INSERT INTO PERSON.PERSON_TYPE(PERSON_TYPEID, NAME) VALUES( 8, '广告部经理
');
INSERT INTO PERSON.PERSON_TYPE(PERSON_TYPEID, NAME) VALUES( 9, '财务部经理
');
```

插入结果如表 5.6.3 所示：

表 5.6.3

PERSON_TYPEID	NAME

1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	销售总监
6	人力资源部经理
8	广告部经理
9	财务部经理

4) 不允许用户修改自增列的值。

```
UPDATE  PERSON.PERSON_TYPE SET PERSON_TYPEID = 9 WHERE NAME = '广告部经理';
```

修改失败。对于自增列，不允许 UPDATE 操作。

5) 还原 IDENTITY_INSERT 属性。

```
SET IDENTITY_INSERT PERSON.PERSON_TYPE OFF;
```

6) 插入后再次查询。注意观察自增列当前值的变化。

```
INSERT  INTO  PERSON.PERSON_TYPE(NAME)  VALUES('市场总监');
```

表 5.6.4

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	销售总监
6	人力资源部经理
8	广告部经理
9	财务部经理
10	市场总监

第 6 章 视图

视图是从一个或几个基表(或视图)导出的表，它是一个虚表，即数据字典中只存放视图的定义(由视图名和查询语句组成)，而不存放对应的数据，这些数据仍存放在原来的基表中。当需要使用视图时，则执行其对应的查询语句，所导出的结果即为视图的数据。因此当基表中的数据发生变化时，从视图中查询出的数据也随之改变了，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据和变化。由此可见，视图是关系数据库系统提供给用户以多种角度观察数据库中数据的重要机制，体现了数据库本身最重要的特色和功能，它简化了用户数据模型，提供了逻辑数据独立性，实现了数据共享和数据的安全保密。视图是数据库技术中一个十分重要的功能。

视图一经定义，就可以和基表一样被查询、修改和删除，也可以在视图之上再建新视图。由于对视图数据的更新均要落实到基表上，因而操作起来有一些限制，读者应注意如何才能在视图中正确更新数据。在本章各例中，如不特别说明，以下例子均使用 BOOKSHOP 示例库，用户均为建表者 SYSDBA。

6.1 视图的作用

视图是提供给用户以多种角度观察数据库中数据的重要机制。尽管在对视图作查询和更新时有各种限制，但只要用户对 DM_SQL 语言熟悉，合理使用视图对用户建立自己的管理信息系统会带来很多的好处和方便，归纳起来，主要有以下几点：

1. 用户能通过不同的视图以多种角度观察同一数据

可针对不同需要建立相应视图，使他们从不同的需要来观察同一数据库中的数据。

2. 简化了用户操作

由于视图是从用户的实际需要中抽取出来的虚表，因而从用户角度来观察这种数据库结构必然简单清晰。另外，由于复杂的条件查询已在视图定义中一次给定，用户再对该视图查询时也简单方便得多了。

3. 为需要隐蔽的数据提供了自动安全保护

所谓“隐蔽的数据”是指通过某视图不可见的数据。由于对不同用户可定义不同的视图，使需要隐蔽的数据不出现在不应该看到这些数据的用户视图上，从而由视图机制自动提供了对机密数据的安全保密功能。

4. 为重构数据库提供了一定程度的逻辑独立性

在建立调试和维护管理信息系统的过程中，由于用户需求的变化、信息量的增长等原因，经常会出现数据库的结构发生变化，如增加新的基表，或在已建好的基表中增加新的列，或需要将一个基表分解成两个子表等，这称为数据库重构。数据的逻辑独立性是指当数据库重构时，对现有用户和用户程序不产生任何影响。

在管理信息系统运行过程中，重构数据库最典型的示例是将一个基表垂直分割成多个表。将经常要访问的列放在速度快的服务器上，而不经常访问的列放在较慢的服务器上。

例如将 PRODUCT 表分解成两个基表。

```
PRODUCT(PRODUCTID,NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCT_CATEGORYID,PRODUCTNO,DESCRIPTION,PHOTO,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME),
```

分解为两个基表：

```
PRODUCT_1(PRODUCTID,NAME,AUTHOR,PUBLISHER,NOWPRICE)
PRODUCT_2(PRODUCTID,PUBLISHERTIME,PRODUCT_CATEGORYID,PRODUCTNO,DESCRIPTION,PHOTO,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
```

并将 PRODUCT 表中的数据分别插入这两个新建表中，再删去 PRODUCT 表。这样一来，原有用户程序中有 PRODUCT 表的操作就均无法进行了。为了减少对用户程序影响，这时可在 PRODUCT_1 和 PRODUCT_2 两基表上建立一个名字为 PRODUCT 的视图，因为新建视图维护了用户外模式的原状，用户的应用程序不用修改仍可通过视图查询到数据，从而较好支持了数据的逻辑独立性。

6.2 视图的定义

语法格式

```
CREATE [OR REPLACE] VIEW
[<模式名>.]<视图名>[(<列名> {,<列名>})]
AS <查询说明>
[WITH [LOCAL|CASCADED]CHECK OPTION]|[WITH READ ONLY];
<查询说明>::=<表查询> | <表连接>
<表查询>::=<子查询表达式>[ORDER BY 子句]
```

参数

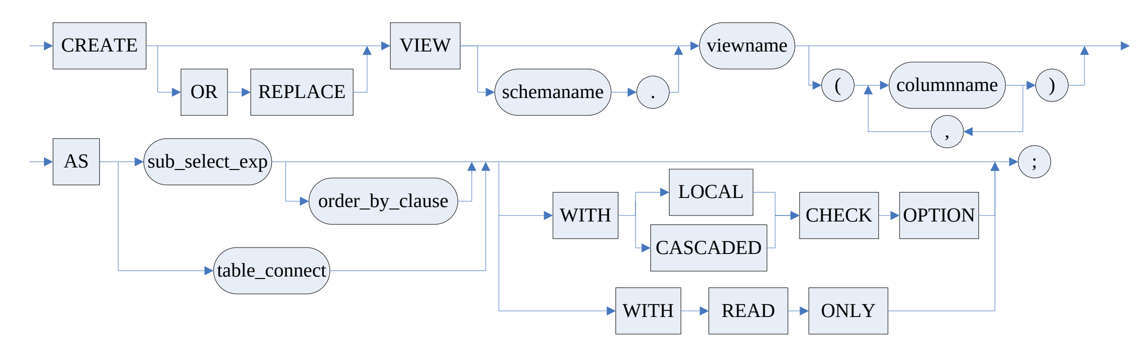
1. <模式名> 指明被创建的视图属于哪个模式，缺省为当前模式；
2. <视图名> 指明被创建的视图的名称；
3. <列名> 指明被创建的视图中列的名称；
4. <子查询表达式> 标识视图所基于的表的行和列。其语法遵照 SELECT 语句的语法规则；
5. <表连接> 请参看第四章连接查询部分；
6. WITH CHECK OPTION 此选项用于可更新视图中。指明往该视图中 insert 或 update 数据时，插入行或更新行的数据必须满足视图定义中<查询说明>所指定的条件。如果不带该选项，则插入行或更新行的数据不必满足视图定义中<查询说明>所指定的条件；

[LOCAL|CASCADED] 用于当前视图是根据另一个视图定义的情况。当通过视图向基表中 insert 或 update 数据时，LOCAL|CASCADED 决定了满足 CHECK 条件的范围。指定 LOCAL，要求数据必须满足当前视图定义中<查询说明>所指定的条件；指定 CASCADED，数据必须满足当前视图，以及所有相关视图定义中<查询说明>所指定的条件。

MPP 系统下不支持该 WITH CHECK OPTION 操作。

7. WITH READ ONLY 指明该视图是只读视图，只可以查询，但不可以做其他 DML 操作；如果不带该选项，则根据 DM 自身判断视图是否可更新的规则判断视图是否只读。

图例



语句功能

供 DBA 或该视图的拥有者且具有 CREATE VIEW 权限的用户定义视图。

使用说明

1．<视图名>后所带<列名>不得同名，个数必须与<查询说明>中 SELECT 后的<值表达式>的个数相等。如果<视图名>后不带<列名>，则隐含该视图中的列由<查询说明>中 SELECT 后的各<值表达式>组成，但这些<值表达式>必须是单纯列名。如果出现以下三种情况之一，<视图名>后的<列名>不能省：

1） <查询说明>中 SELECT 后的<值表达式>不是单纯的列名，而包含集函数或运算表达式；

2） <查询说明>包含了多表连接，使得 SELECT 后出现了几个不同表中的同名列作为视图列；

3） 需要在视图中为某列取与<查询说明>中 SELECT 后<列名>不同的名字。

最后要强调的是：<视图名>后的<列名>必须全部省略或全部写明。

2．为了防止用户通过视图更新基表数据时，无意或故意更新了不属于视图范围内的基表数据，在视图定义语句的子查询后提供了可选项 WITH CHECK OPTION。如选择，表示往该视图中插入或修改数据时，要保证插入行或更新行的数据满足视图定义中<查询说明>所指定的条件，不选则可不满足；

4．视图是一个逻辑表，它自己不包含任何数据；

5．视图上可以建立 INSTEAD OF 触发器（只允许行级触发），但不允许创建 BEFORE/AFTER 触发器；

6．视图分为可更新视图和不可更新视图，具体规则参见 6.6 节。

权限

该语句的使用者必须对<查询说明>中的每个表均具有 SELECT 权限。

举例说明

例 对 VENDOR 表创建一个视图，名为 VENDOR_EXCELLENT，保存信誉等级为 1 的供应商，列名有：VENDORID，ACCOUNTNO，NAME，ACTIVEFLAG。

```
CREATE  VIEW  PURCHASING.VENDOR_EXCELLENT  AS
SELECT  VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT
FROM  PURCHASING.VENDOR
WHERE  CREDIT = 1;
```

由于视图列名与查询说明中 SELECT 后的列名相同，所以视图名后的列名可省。

运行该语句，AS 后的查询语句并未执行，系统只是将所定义的<视图名>及<查询说明>送数据字典保存。对用户来说，就像在数据库中已经有 VENDOR_EXCELLENT 这样一个表。

如果对该视图作查询：

```
SELECT  *  FROM  PURCHASING.VENDOR_EXCELLENT;
```

查询结果见下表 6.1.1。

表 6.1.1

VENDORID	ACCOUNTNO	NAME	ACTIVEFLAG	CREDIT
3	00	北京十月文艺出版社	1	1
4	00	人民邮电出版社	1	1
5	00	清华大学出版社	1	1
6	00	中华书局	1	1
7	00	广州出版社	1	1
8	00	上海出版社	1	1
9	00	21 世纪出版社	1	1
10	00	外语教学与研究出版社	1	1
11	00	社械工业出版社	1	1
12	00	文学出版社	1	1

用户可以在该表上作数据库的查询、插入、删除、修改等操作。在建好的视图之上还可以再建立视图。

由于以上定义包含可选项 WITH CHECK OPTION，以后对该视图作插入、修改和删除操作时，系统均会自动用 WHERE 后的条件作检查，不满足条件的数据，则不能通过该视图更新相应基表中的数据。

例 视图也可以建立在多个基表之上。构造一视图，名为 SALESPERSON_INFO，用来保存销售人员的信息，列名有：SALESPERSONID，TITLE，NAME，SALESLASTYEAR。

```
CREATE  VIEW  SALES.SALESPERSON_INFO  AS
SELECT  T1.SALESPERSONID,  T2.TITLE,  T3.NAME,  T1.SALESLASTYEAR
FROM    SALES.SALESPERSON  T1,  RESOURCES.EMPLOYEE  T2,  PERSON.PERSON  T3
WHERE   T1.EMPLOYEEID = T2.EMPLOYEEID  AND  T2.PERSONID = T3.PERSONID;
```

如果对该视图作查询：

```
SELECT  *  FROM  SALES.SALESPERSON_INFO;
```

查询结果见下表 6.1.2。

表 6.1.2

SALESPERSONID	TITLE	NAME	SALESLASTYEAR
1	销售代表	郭艳	10.0000
2	销售代表	孙丽	20.0000

由前面的介绍可知，基表中的数据均是基本数据。为了减少数据冗余，由基本数据经各种计算统计出的数据一般是不存贮的，但这样的数据往往又要经常使用，这时可将它们定义成视图中的数据。

例 在 PRODUCT_VENDOR 上建立一视图，用于统计数量。

```
CREATE  VIEW  PRODUCTION.VENDOR_STATIS(VENDORID,  PRODUCT_COUNT)  AS
SELECT  VENDORID,  COUNT(PRODUCTID)
FROM   PRODUCTION.PRODUCT_VENDOR
GROUP  BY  VENDORID
ORDER  BY  VENDORID;
```

在该语句中，由于 SELECT 后出现了集函数 COUNT(PRODUCTID)，不属于单纯的列名，所以视图中的对应列必须重新命名，即在<视图名>后明确说明视图的各个列名。

由于该语句中使用了 GROUP BY 子句，所定义的视图也称分组视图。分组视图的<视图名>后所带<列名>不得包含集函数。

如果对该视图作查询：

```
SELECT  *  FROM  PRODUCTION.VENDOR_STATIS;
```

查询结果如下表 6.1.3 所示。

表 6.1.3

VENDORID	PRODUCT_COUNT

5	1
6	2
7	1
8	1
9	1
10	1
11	1

6.3 视图的删除

一个视图本质上是基于其他基表或视图上的查询，我们把这种对象间关系称为依赖。用户在创建视图成功后，系统还隐式地建立了相应对象间的依赖关系。在一般情况下，当一个视图不被其他对象依赖时可以随时删除视图。

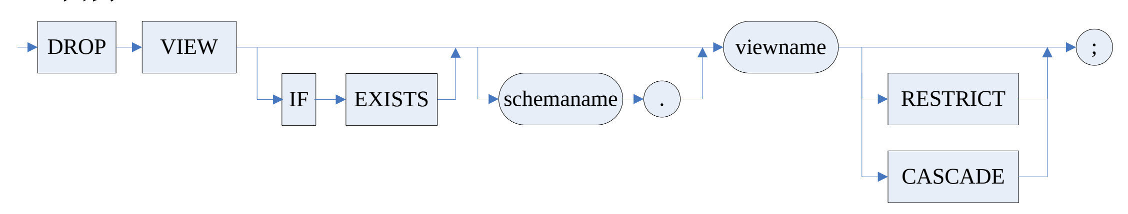
语法格式

```
DROP VIEW [IF EXISTS][<模式名>.]<视图名> [RESTRICT | CASCADE];
```

参数

1. <模式名>　　指明被删除视图所属的模式，缺省为当前模式；
2. <视图名>　　指明被删除视图的名称。

图例



使用说明

1．删除不存在的视图会报错。若指定IF EXISTS关键字后，删除不存在的视图，不会报错；

2．视图删除有两种方式：RESTRICT/CASCADE 方式。其中 RESTRICT 为缺省值。当设置 dm.ini 中的参数 DROP_CASCADE_VIEW 值为 1 时，如果在该视图上建有其它视图，必须使用 CASCADE 参数才可以删除所有建立在该视图上的视图，否则删除视图的操作不会成功；当设置 dm.ini 中的参数 DROP_CASCADE_VIEW 值为 0 时，RESTRICT 和 CASCADE 方式都会成功，且只会删除当前视图，不会删除建立在该视图上的视图；

3．如果没有删除参考视图的权限，那么两个视图都不会被删除；

4．该视图删除后，用户在其上的权限也均自动取消，以后系统中再建的同名视图，是与他毫无关系的视图。

权限

使用者必须拥有 DBA 权限或是该视图的拥有者。

举例说明

例 删除视图 VENDOR_EXCELLENT，可使用下面的语句：

```
DROP  VIEW  PURCHASING.VENDOR_EXCELLENT;
```

当该视图对象被其他对象依赖时，用户在删除视图时必须带 CASCADE 参数，系统会将依赖于该视图的其他数据库对象一并删除，以保证数据库的完整性。

例 删除视图 SALES.SALESPERSON_INFO，同时删除此视图上的其他视图，可使用下

面的语句：

```
DROP  VIEW  SALES.SALESPERSON_INFO CASCADE;
```

6.4 视图的查询

视图一旦定义成功，对基表的所有查询操作都可用于视图。对于用户来说，视图和基表在进行查询操作时没有区别。

例 从 VENDOR_EXCELLENT 中查询 ACTIVEFLAG 为 1 的供应商的编号和名称。

```
SELECT  VENDORID, NAME
FROM  PURCHASING.VENDOR_EXCELLENT
WHERE  ACTIVEFLAG = 1;
```

系统执行该语句时，先从数据字典中取出视图 VENDOR_EXCELLENT 的定义，按定义语句查询基表，得到视图表，再根据条件:ACTIVEFLAG = 1 查询视图表，选择所需列名，得到结果如下表 6.3.1 所示。

表 6.3.1

VENDORID	NAME
3	北京十月文艺出版社
4	人民邮电出版社
5	清华大学出版社
6	中华书局
7	广州出版社
8	上海出版社
9	21 世纪出版社
10	外语教学与研究出版社
11	社械工业出版社
12	文学出版社

视图尽管是虚表，但它仍可与其它基表或视图作连接查询，也可以出现在子查询中。

例 查询信誉等级为 1 的供应商供应的图书编号、名称、通常价格和供应商名称。

```
SELECT  T1.PRODUCTID, T1.NAME, T2.STANDARDPRICE, T3.NAME
FROM  PRODUCTION.PRODUCT  T1,  PRODUCTION.PRODUCT_VENDOR  T2,
       PURCHASING.VENDOR_EXCELLENT  T3
WHERE  T1.PRODUCTID  =  T2.PRODUCTID  AND T2.VENDORID = T3.VENDORID;
```

系统执行该语句时，先从数据字典中取出视图 VENDOR_EXCELLENT 的定义，按定义语句查询基表，得到视图表，再将 PRODUCT、PRODUCT_VENDOR 和视图表按连接条件作连接，选择所需列名，得到最后结果。

得到结果如下表 6.3.2 所示。

表 6.3.2

PRODUCTID	NAME	STANDARDPRICE	NAME
1	红楼梦	25.0000	中华书局
2	水浒传	25.0000	中华书局
3	老人与海	25.0000	上海出版社
4	射雕英雄传(全四册)	25.0000	广州出版社
7	数据结构(C 语言版)(附光盘)	25.0000	清华大学出版社
9	突破英文基础词汇	25.0000	外语教学与研究出版社
10	噼里啪啦丛书(全 7 册)	25.0000	21 世纪出版社

11	工作中无小事	25.0000	机械工业出版社

6.5 视图的编译

一个视图依赖于其基表或视图，如果基表定义发生改变，如增删一列，或者视图的相关权限发生改变，可能导致视图无法使用。在这种情况下，可对视图重新编译，检查视图的合法性。

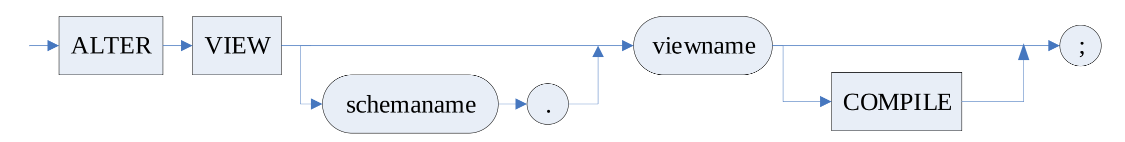
语法格式

```
ALTER VIEW [<模式名>.]<视图名> COMPILE;
```

参数

1. <模式名>　　指明被编译视图所属的模式，缺省为当前模式；
2. <视图名>　　指明被编译视图的名称；

图例



使用说明

对视图的定义重新进行分析和编译，如果编译出错，则报错，可以此判断视图依赖的基表是否已被删除或修改了表定义。

权限

使用者必须拥有 DBA 权限或是该视图的拥有者。

举例说明

例 重新编译视图 PURCHASING.VENDOR_EXCELLENT。

```
ALTER VIEW PURCHASING.VENDOR_EXCELLENT COMPILE;
```

6.6 视图数据的更新

视图数据的更新包括插入(INSERT)、删除(DELETE)和修改(UPDATE)三类操作。由于视图是虚表，并没有实际存放数据，因此对视图的更新操作均要转换成对基表的操作。在 SQL 语言中，对视图数据的更新语句与对基表数据的更新语句在格式与功能方面是一致的。

例 从视图 VENDOR_EXCELLENT 中将名称为人民邮电出版社的 ACTIVEFLAG 改为 0。

```
UPDATE  PURCHASING.VENDOR_EXCELLENT
SET  ACTIVEFLAG = 0 WHERE NAME = '人民邮电出版社';
```

系统执行该语句时，首先从数据字典中取出视图 VENDOR_EXCELLENT 的定义，将其中的查询说明与对视图的修改语句结合起来，转换成对基表的修改语句，然后再执行这个转换后的更新语句。

```
UPDATE  PURCHASING.VENDOR
SET  ACTIVEFLAG = 0
WHERE  NAME = '人民邮电出版社'  AND  CREDIT = 1;
```

例 往视图 VENDOR_EXCELLENT 中插入一个新的记录，其中 ACCOUNTNO 为 00，NAME 为电子工业出版社，ACTIVEFLAG 为 1，WEBURL 为空。则相应的插入语句为：

```
INSERT  INTO  PURCHASING.VENDOR_EXCELLENT(ACCOUNTNO, NAME, ACTIVEFLAG,
CREDIT) VALUES('00 ', '电子工业出版社', 1, 1);
```

例 从视图 VENDOR_EXCELLENT 中删除名称为中华书局的供应商信息。

```
DELETE  FROM  PURCHASING.VENDOR_EXCELLENT  WHERE  NAME = '中华书局';
```

系统将该语句与 VENDOR_EXCELLENT 视图的定义相结合，转换成对基表的语句：

```
DELETE  FROM  PURCHASING.VENDOR WHERE  NAME = '中华书局'  AND  CREDIT = 1;
```

系统执行该语句，会报告违反约束错误，因为 VENDOR_EXCELLENT 尽管是视图，在做更新时一样要考虑基表间的引用关系。PRODUCT_VENDOR 表与 VENDOR 表存在着引用关系，PRODUCT_VENDOR 表为引用表，VENDOR 表为被引用表，只有当引用表中没有相应 VENDORID 时才能删除 VENDOR 表中相应记录。

在关系数据库中，并不是所有视图都是可更新的，即并不是所有的视图更新语句均能有意义地转换成相应的基表更新语句，有些甚至是根本不能转换。例如对视图 VENDOR_STATIS：

```
UPDATE  PRODUCTION.VENDOR_STATIS
SET  PRODUCT_COUNT  = 3
WHERE VENDORID = 5;
```

由于产品数量是查询结果按供应商编号分组后各组所包含的行数，这是无法修改的。像这样的视图为不可更新视图。

目前，不同的关系数据库管理系统产品对更新视图的可操作程度均有差异。DM 系统有这样的规定：

1. 如果视图建在单个基表或单个可更新视图上，且该视图包含了表中的全部聚集索引键，则该视图为可更新视图；
2. 如果视图由两个以上的基表导出时，则该视图不允许更新；
3. 如果视图列是集函数，或视图定义中的查询说明包含集合运算符、GROUP BY 子句或 HAVING 子句，则该视图不允许更新；
4. 在不允许更新视图之上建立的视图也不允许更新。

应该说明的是：只有当视图是可更新的时候，才可以选择 WITH CHECK OPTION 项。

第 7 章 物化视图

物化视图是从一个或几个基表导出的表，同视图相比，它存储了导出表的真实数据。当基表中的数据发生变化时，物化视图所存储的数据将变得陈旧，用户可以通过手动刷新或自动刷新来对数据进行同步。

在本章各例中，如不特别说明，以下例子用户均为建表者 SYSDBA。

7.1 物化视图的定义

语法格式

```
CREATE MATERIALIZED VIEW [<模式名>.]<物化视图名>[(<列名>{,<列名>})][BUILD IMMEDIATE|BUILD DEFERRED][<STORAGE 子句>][<物化视图刷新选项>][<查询改写选项>]AS<查询说明>
<STORAGE 子句>::=参见 3.6.1.1 节
<查询说明>::= <表查询> | <表连接>
<表查询>::=<子查询表达式>[ORDER BY 子句]
<物化视图刷新选项> ::= REFRESH <刷新选项> {<刷新选项>} | NEVER REFRESH
<刷新选项> ::= [FAST | COMPLETE | FORCE] [ON DEMAND | ON COMMIT ] [START WITH datetime_expr | NEXT datetime_expr] [WITH PRIMARY KEY | WITH ROWID]
<查询改写选项>::= [DISABLE | ENABLE] QUERY REWRITE
<datetime_expr>::= SYSDATE[+数值常量]
```

参数

1. <模式名> 指明被创建的视图属于哪个模式，缺省为当前模式；
2. <物化视图名> 指明被创建的物化视图的名称；
3. <列名> 指明被创建的物化视图中列的名称；
4. [BUILD IMMEDIATE|BUILD DEFERRED] 指明 BUILD IMMEDIATE 为立即填充数据，默认为立即填充；BUILD DEFERRED 为延迟填充，使用这种方式要求第一次刷新必须为 COMPLETE 完全刷新。
5. <子查询表达式> 标识物化视图所基于的表的行和列。其语法遵照 SELECT 语句的语法规则；
6. <表连接> 请参看第四章表连接查询部分；
7. 定义查询中的 ORDER BY 子句仅在创建物化视图时使用，此后 ORDER BY 被忽略；
8. 刷新模式
   - FAST

     根据相关表上的数据更改记录进行增量刷新。普通 DML 操作生成的记录存在于物化视图日志。使用 FAST 刷新之前，必须先建好物化视图日志。
   - COMPLETE

     通过执行物化视图的定义脚本进行完全刷新。
   - FORCE

     默认选项。当快速刷新可用时采用快速刷新，否则采用完全刷新。
9. 刷新时机
   - ON COMMIT

     在相关表上视图提交时进行快速刷新。刷新是由异步线程执行的，因此 COMMIT 执行结束后可能需要等待一段时间物化视图数据才是最新的。

约束：

含有对象类型的不支持；

包含远程表的不支持。

- START WITH … NEXT

  START WITH 用于指定首次刷新物化视图的时间，NEXT 指定自动刷新的间隔；
  如果省略 START WITH 则首次刷新时间为当前时间加上 NEXT 指定的间隔；
  如果指定 START WITH 省略 NEXT 则物化视图只会刷新一次；
  如果二者都未指定物化视图不会自动刷新。

- ON DEMAND

  由用户通过 REFRESH 语法进行手动刷新。如果指定了 START WITH 和 NEXT 子句就没有必要指定 ON DEMAND。

- NEVER REFRESH

  物化视图从不进行刷新。可以通过 ALTER MATERALIZED VIEW <物化视图名> FRESH 进行更改。

10. 刷新选项

- WITH PRIMARY KEY

  默认选项。

  - 只能基于单表
  - 必须含有 PRIMARY KEY 约束，选择列必须直接含有所有的 PRIMARY KEY（UPPER(col_name)的形式不可接受）
  - 不能含有对象类型

- WITH ROWID

  - 只能基于单表
  - 不能含有对象类型
  - 如果使用 WITH ROWID 的同时使用快速刷新，则必须将 ROWID 提取出来，和其他列名一起，以别名的形式显示

11. QUERY REWRITE 选项

- ENABLE

允许物化视图用于查询改写。

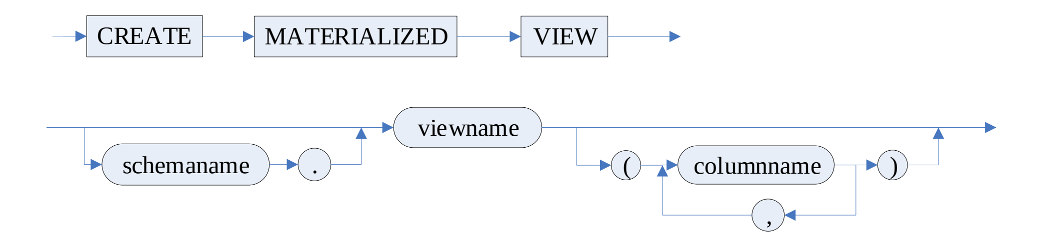
- DISABLE

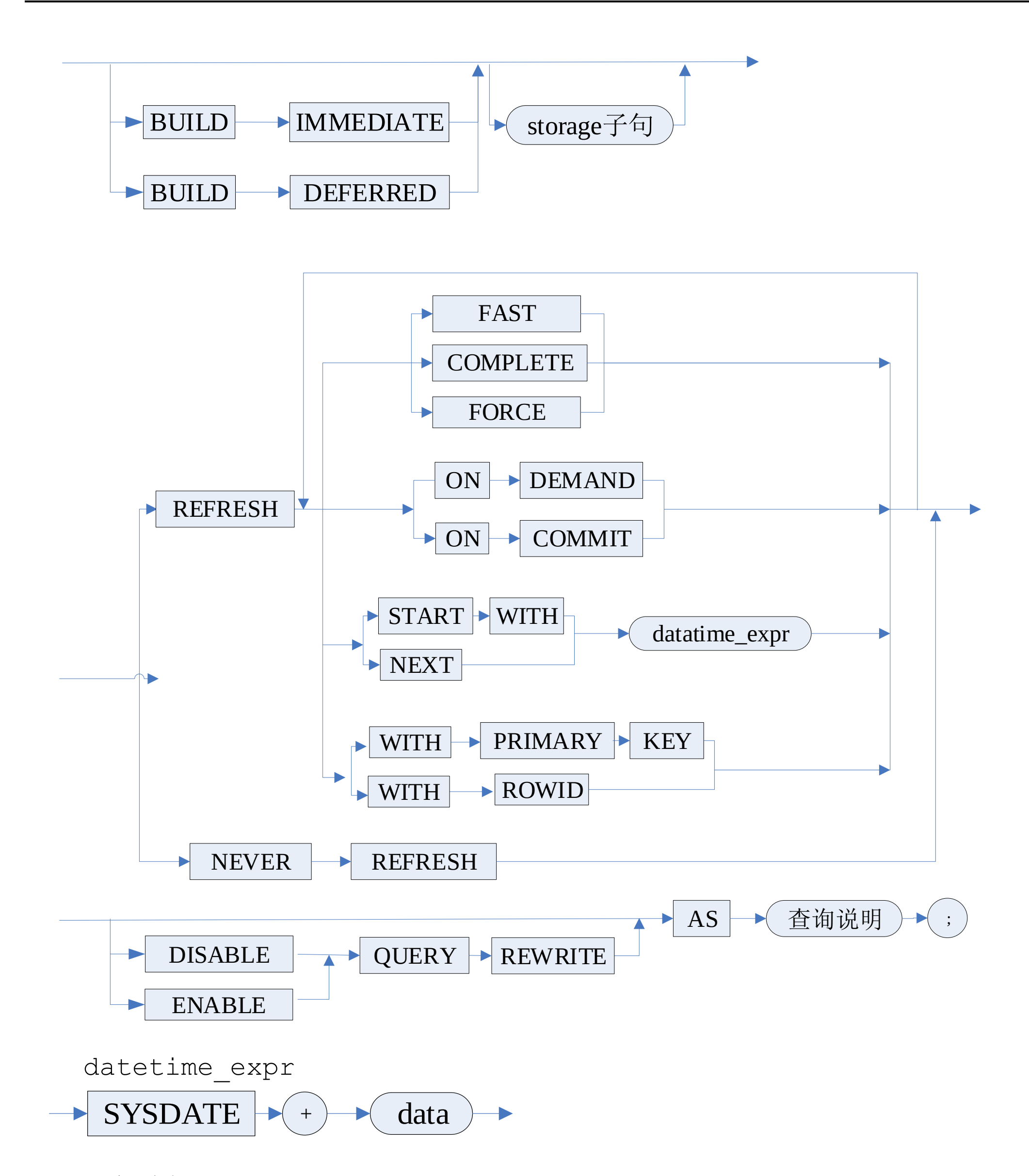
禁止物化视图用于查询改写。

目前 DM8 仅语法支持查询改写选项，实际功能未实现。

12. datetime_expr 只能是日期常量表达式，SYSDATE[+数值常量]或日期间隔；
13. 如果物化视图中包含大字段列，需要用户手动指定 STORAGE(USING LONG ROW)的存储方式。

图例





语句功能

供 DBA 或该物化视图的拥有者且具有 CREATE MATERIALIZED VIEW 权限的用户定义物化视图。

使用说明

1．创建物化视图时，会产生两个字典对象：物化视图和物化视图表，后者用于存放真实的数据；

2．快速刷新物化视图的限制见本章第 8 节《物化视图的限制》；

3．由于受物化视图表的命名规则所限，物化视图名称长度必须小于 123 个字节；

权限

1．在自己模式下创建物化视图时，该语句的使用者必须被授予 CREATE MATERIALIZED VIEW 系统权限，且至少拥有 CREATE TABLE 或者 CREATE ANY TABLE 两个系统权限中的一个；

2．在其他用户模式下创建物化视图时，该语句的使用者必须具有 CREATE ANY MATERIALIZED VIEW 系统权限，且物化视图的拥有者必须拥有 CREATE TABLE 系统权限；

3．物化视图的拥有者必须对<查询说明>中的每个表均具有 SELECT 权限或者具有

SELECT ANY TABLE 系统权限。

举例说明

例 对 VENDOR 表创建一个物化视图，名为 MV_VENDOR_EXCELLENT，保存信誉等级为 1 的供应商，列名有：VENDORID、ACCOUNTNO、NAME、ACTIVEFLAG、CREDIT。不允许查询改写，依据 ROWID 刷新且刷新间隔为一天。

```
CREATE MATERIALIZED VIEW  PURCHASING.MV_VENDOR_EXCELLENT
REFRESH WITH ROWID START WITH SYSDATE NEXT SYSDATE + 1 AS
SELECT  VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT
FROM  PURCHASING.VENDOR
WHERE  CREDIT = 1;
```

如果使用 WITH ROWID 的同时，后面还要使用快速刷新，则此处的语句应写为：

```
--先创建好物化视图，创建步骤参考 7.6 节。
CREATE MATERIALIZED VIEW  PURCHASING.MV_VENDOR_EXCELLENT
REFRESH FAST WITH ROWID START WITH SYSDATE NEXT SYSDATE + 1 AS
SELECT  VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT,ROWID AS X
FROM  PURCHASING.VENDOR
WHERE  CREDIT = 1;
```

运行该语句后，DM 服务器将得到：1）物化视图：MV_VENDOR_EXCELLENT；2）物化视图表：MTAB$_MV_VENDOR_EXCELLENT；3）定时刷新的物化视图触发器：MTRG_REFRESH_MVIEW_1670 （假定 MTAB$_MV_VENDOR_EXCELLENT 对象的 ID 是 1670）。

对该物化视图进行查询：

```
SELECT  *  FROM  PURCHASING.MV_VENDOR_EXCELLENT;
```

查询结果见下表 7.1.1。

表 7.1.1

VENDORID	ACCOUNTNO	NAME	ACTIVEFLAG	CREDIT
3	00	北京十月文艺出版社	1	1
4	00	人民邮电出版社	1	1
5	00	清华大学出版社	1	1
6	00	中华书局	1	1
7	00	广州出版社	1	1
8	00	上海出版社	1	1
9	00	21 世纪出版社	1	1
10	00	外语教学与研究出版社	1	1
11	00	社械工业出版社	1	1
12	00	文学出版社	1	1

7.2 物化视图的修改

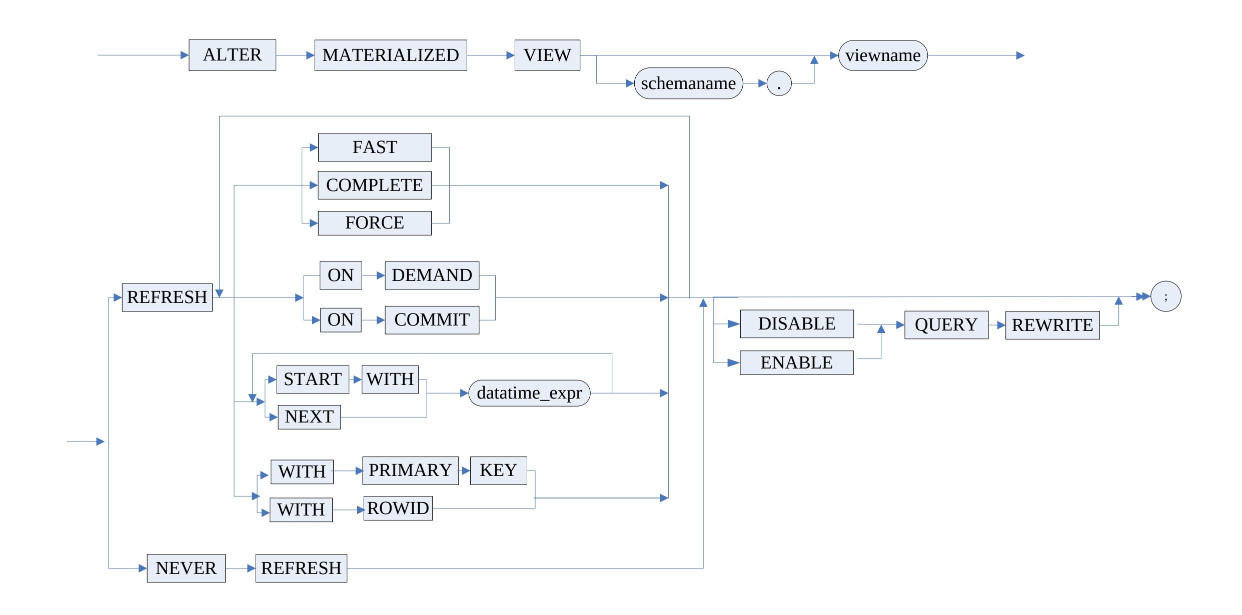
语法格式

```
ALTER MATERIALIZED VIEW [<模式名>.]<物化视图名>
[<物化视图刷新选项>]
[<查询改写选项>]
```

参数

<物化视图刷新选项>和<查询改写选项>参见本章第 1 节"物化视图的定义"。

图例



权限

使用者必须是该物化视图的拥有者或者拥有 ALTER ANY MATERIALIZED VIEW 系统权限。

举例说明

例 修改物化视图 MV_VENDOR_EXCELLENT，使之可以用于查询改写。

```
ALTER MATERIALIZED VIEW PURCHASING.MV_VENDOR_EXCELLENT ENABLE QUERY REWRITE;
```

例 修改物化视图 MV_VENDOR_EXCELLENT 为完全刷新。

```
ALTER MATERIALIZED VIEW PURCHASING.MV_VENDOR_EXCELLENT REFRESH COMPLETE;
```

7.3 物化视图的删除

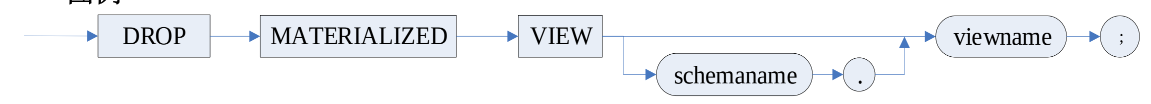
语法格式

```
DROP MATERIALIZED VIEW [<模式名>.]<物化视图名>;
```

参数

1. <模式名> 指明被删除视图所属的模式，缺省为当前模式；
2. <物化视图名> 指明被删除物化视图的名称；

图例



使用说明

1. 物化视图删除时会清除物化视图和物化视图表；
2. 物化视图删除后，用户在其上的权限也均自动取消，以后系统中再建的同名物化视图，是与它毫无关系的物化视图；
3. 用户不能直接删除物化视图表对象。

权限

使用者必须是物化视图的拥有者或者拥有 DROP ANY MATERIALIZED VIEW 系统权限。

举例说明

例 删除物化视图 MV_VENDOR_EXCELLENT，可使用下面的语句：

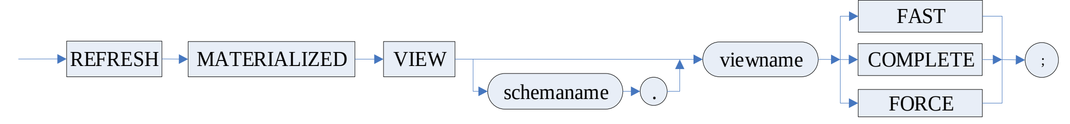
```
DROP MATERIALIZED VIEW PURCHASING.MV_VENDOR_EXCELLENT;
```

7.4 物化视图的更新

语法格式

```
REFRESH MATERIALIZED VIEW  [<模式名>.] <物化视图名>
[FAST|
COMPLETE|
FORCE]
```

图例



权限

1. 如果是基于物化视图日志的更新，则使用者必须是物化视图日志的拥有者或者具有 SELECT ANY TABLE 系统权限；

2. 使用者必须是物化视图的拥有者或者具有 SELECT ANY TABLE 系统权限。

使用说明

物化视图的更新语句总是自动提交的，不能回滚。

举例说明

例 采用 FAST 方式刷新物化视图 MV_VENDOR_EXCELLENT。

```
REFRESH MATERIALIZED VIEW  PURCHASING.MV_VENDOR_EXCELLENT FAST; --使用快速刷新前，必须先建好物化视图日志
```

7.5 物化视图允许的操作

对物化视图进行查询或建立索引时这两种操作都会转为对其物化视图表的处理。用户不能直接对物化视图及物化视图表进行插入、删除、更新和 TRUNCATE 操作，对物化视图数据的修改只能通过刷新物化视图语句进行。

7.6 物化视图日志的定义

物化视图的快速刷新依赖于基表上的物化视图日志，物化视图日志记录了基表的变化信息。

语法格式

```
CREATE MATERIALIZED VIEW  LOG ON [<模式名>.]<表名>
[<STORAGE 子句>][<WITH 子句>][<PURGE 选项>]
<WITH 子句>::= WITH { PRIMARY KEY| ROWID | SEQUENCE | (<列名> {, <列名>})}
<PURGE 选项>::= PURGE IMMEDIATE [ SYNCHRONOUS | ASYNCHRONOUS ]
        | PURGE START WITH datetime_expr [ NEXT datetime_expr | REPEAT INTERVAL interval_expr]
```

参数

1. <模式名> 指明物化视图日志基表所属的模式，缺省为当前模式；

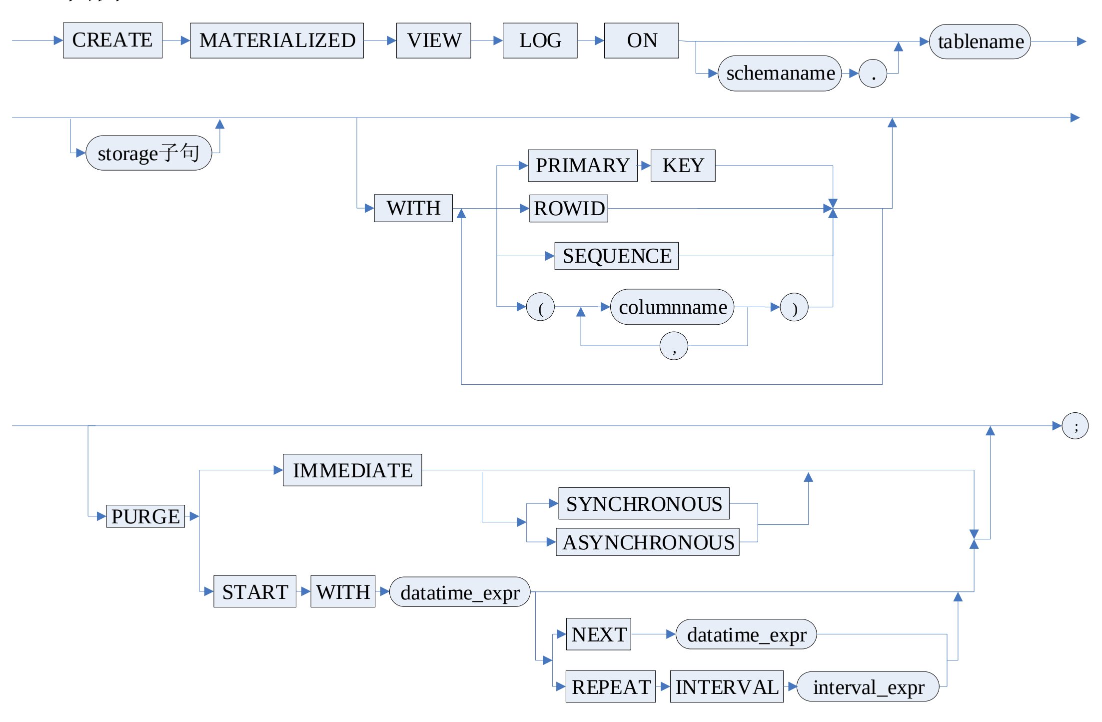
2. <表名> 指明创建日志的基表；

3. <WITH 子句> 基表中的哪些列将被包含到物化视图日志中，SEQUENCE 表示物

化视图日志表中有 SEQUENCE$$唯一标识列，SEQUENCE 为默认选项；

4. <PURGE 选项> 指定每隔多长时间对物化视图日志中无用的记录进行一次清除。分两种情况：一是 IMMEDIATE 立即清除；二是 START WITH 定时清除。缺省是 PURGE IMMEDIATE。SYNCHRONOUS 为同步清除；ASYNCHRONOUS 为异步清除。ASYNCHRONOUS 和 SYNCHRONOUS 的区别是前者新开启一个事务来进行日志表的清理，后者是在同一个事务里。目前，ASYNCHRONOUS 仅语法支持，功能未实现；
5. datetime_expr 只能是日期常量表达式，SYSDATE[+数值常量]或日期间隔；
6. 与物化视图可能依赖多个基表不同，物化视图日志只对应一个基表，因此物化视图日志是否使用行外大字段存储与基表保持一致。

图例



使用说明

1. 在表 T 上创建物化视图日志后会生成：1）MLOG$_T 的日志表；2）MTRG$_T 的表级触发器；3）定时 purge 物化视图日志的触发器：MTRG_PURGE_MVLOG_1270 （假定 MLOG$_T 对象的 ID 是 1270）。用户可以对 MLOG$_T 进行查询但是不能进行插入、删除和更新，触发器由系统维护，用户无法修改删除；

2. 由于物化视图日志表的命名规则所限，日志基表名称长度必须小于 123 个字符；

3. 如果在物化视图 MV 上创建物化视图日志，系统会自动转为在物化视图的基表 MTAB$_MV 上创建物化视图日志，因此会生成 MLOG$_MTAB$_MV 的日志表和 MTRG$_MTAB$_MV 的表级触发器，且要求物化视图名称必须小于 116 个字符，其余限制和普通表一致。

权限

1. 如果是物化视图日志基表的拥有者，使用者必须拥有 CREATE TABLE 系统权限；

2. 如果物化视图日志基表是其它模式下的表，使用者必须拥有 CREATE ANY TABLE 系统权限，且物化视图日志的拥有者必须对<查询说明>中的每个表均具有 SELECT 权限或者具有 SELECT ANY TABLE 系统权限；

3. 物化视图日志表仅支持基于的表为普通表、堆表和物化视图。

举例说明

例 在 PURCHASING．VENDOR 上创建物化视图日志，每天定时 PURGE。

```
CREATE MATERIALIZED VIEW LOG ON PURCHASING.VENDOR WITH ROWID(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT) PURGE START WITH SYSDATE + 5 REPEAT INTERVAL '1' DAY;
```

注：间隔一天 PURGE 也可以写成 PURGE NEXT SYSDATE + 1。

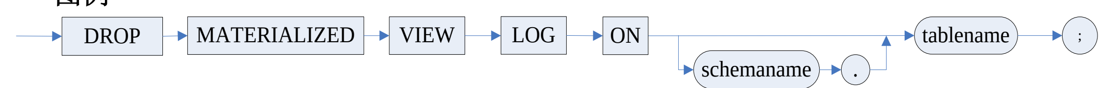
7.7 物化视图日志的删除

语法格式

```
DROP MATERIALIZED VIEW LOG ON [<模式名>.]<表名>
```

物化视图日志删除时会同时 DROP 掉日志表对象和触发器对象。另外，删除物化视图基表的同时，也会级联删除相应的物化视图日志。

图例



权限

使用者必须拥有删除表的权限。

7.8 物化视图的限制

7.8.1 物化视图的一般限制

1. 物化视图定义只能包含用户创建的表、视图和物化视图对象，且不能为临时表和外部表；
2. 对物化视图日志、物化视图只能进行查询和建索引，不支持插入、删除、更新、MERGE INTO 和 TRUNCATE；
3. 同一表上最多允许建立 127 个物化视图；
4. 包含物化视图的普通视图及游标是不能更新的；
5. 如果对某表进行了 TRUNCATE 操作，那么依赖于它的物化视图必须先进行一次完全刷新后才可以使用快速刷新；
6. 如果对某表进行了快速装载操作，那么依赖于它的物化视图必须先进行一次完全刷新后才可以使用快速刷新；
7. 如果对某表进行了与分区子表数据交换操作，那么依赖于它的物化视图必须先进行一次完全刷新后才可以使用快速刷新。

7.8.2 物化视图的分类

依据物化视图定义中查询语句的不同分为以下五种。

1. SIMPLE：无 GROUP BY，无聚集函数，无连接操作；
2. AGGREGATE：仅包含有 GROUP BY 和聚集函数；
3. JOIN：仅包含有多表连接；
4. Sub-Query：仅包含有子查询；
5. COMPLEX：除上述四种外的物化视图类型。

用户可以通过查看系统视图 SYS.USER_MVIEWS 的 MVIEW_TYPE 列来了解所定义物化视图的分类。

7.8.3 快速刷新通用约束

1. 快速刷新物化视图要求每个基表都包含有物化视图日志，并且物化视图日志的创建时间不得晚于物化视图的最后刷新时间；
2. 不能含有不确定性函数，如 SYSDATE 或 ROWNUM；
3. 不能含有大字段类型；
4. 查询项不能含有分析函数；
5. 查询不能含有 HAVING 子句；
6. 不能包含 ANY、ALL 及 NOT EXISTS；
7. 不能含有层次查询；
8. 不能在多个站点含有相关表；
9. 同一张表上最多允许建立 127 个快速刷新的物化视图；
10. 不能含有 UNION，UNION ALL，MINUS 等集合运算；
11. 不能含有子查询；
12. 只能基于普通表（视图，外部表，派生表等不支持）；
13. WITH PRIMAY KEY 时物化视图定义里如果是单表，则日志表里有 PK，如果是多表，则每张表的日志表里都有 PK；WITH ROWID 时物化视图里是单表，则日志表里必须有 ROWID，如果是多表，则每张日志表里都有 ROWID；
14. 对于 WITH ROWID 的快速刷新需要一一选择 ROWID 并给出别名；
15. WITH PRIMAY KEY 刷新时，物化视图定义中必须包含所有其基于的表的 PK 列；
16. 如果日志定义中没有 WITH PRIMARY KEY 而扩展列又包含了，那么 DM 认为这个和建立日志时指定 WITH PRIMARY KEY 效果相同。也就是说，基于这个日志建立 WITH PK 的快速刷新物化视图是允许的；
17. DM8 目前仅支持简单类型和部分连接物化视图的快速刷新。连接物化视图不支持的具体类型是外连接和自然连接；
18. 连接物化视图不支持 GROUP BY 和聚集操作。

7.8.4 物化视图信息查看

用户可以通过系统视图 SYS.USER_MVIEWS 查看系统中所有物化视图的相关信息，视图定义如下：

列名	数据类型	备注
SCHID	INTEGER	模式 ID
MVIEW_NAME	VARCHAR(128)	物化视图名称
QUERY	TEXT	文本信息
QUERY_LEN	INTEGER	文本信息长度，单位：字节
REWRITE_ENABLED	VARCHAR(128)	是否可以被重写。Y 是，N 否
REFRESH_MODE	VARCHAR(128)	刷新模式：DEMAND,COMMIT
REFRESH_METHOD	VARCHAR(128)	刷新方法：COMPLETE,FORCE,FAST,NEVER
MVIEW_TYPE	VARCHAR(128)	快速刷新类型：

		SIMPLE：简单； AGGREGATE：聚合； JOIN：连接； Sub-Query：子查询； COMPLEX：不可快速刷新。
LAST_REFRESH_TYPE	VARCHAR(128)	最后一次刷新的类型
STALENESS	VARCHAR(128)	物化视图状态： UNUSEABLE：物化视图不可用，即从未刷新过； FRESH：物化视图数据是最新的； NEEDS_COMPILE：物化视图数据陈旧； COMPILE_ERROR：化视图解析出错，如基表不存在； NEEDS_FULL_REFRESH：物化视图数据陈旧，需要一次全刷新； UNDEFINED：物化视图包含远程表状态不可知 。
LAST_REFRESH_DATE	DATETIME(6)	最后刷新的日期

第 8 章 函数

在值表达式中，除了可以使用常量、列名、集函数等之外，还可以使用函数作为组成成份。DM 中支持的函数分为数值函数、字符串函数、日期时间函数、空值判断函数、类型转换函数等。在这些函数中，对于字符串类型的参数或返回值，最大支持的长度为 32K。

本手册还给出了 DM 系统函数的详细介绍。下列各表列出了函数的简要说明。在本章各例中，如不特别说明，各例均使用示例库 BOOKSHOP，用户均为建表者 SYSDBA。

表 8.1 数值函数

序号	函数名	功能简要说明
01	ABS(n)	求数值 n 的绝对值
02	ACOS(n)	求数值 n 的反余弦值
03	ASIN(n)	求数值 n 的反正弦值
04	ATAN(n)	求数值 n 的反正切值
05	ATAN2(n1,n2)	求数值 n1/n2 的反正切值
06	CEIL(n)	求大于或等于数值 n 的最小整数
07	CEILING(n)	求大于或等于数值 n 的最小整数，等价于 CEIL(n)
08	COS(n)	求数值 n 的余弦值
09	COSH(n)	求数值 n 的双曲余弦值
10	COT(n)	求数值 n 的余切值
11	DEGREES(n)	求弧度 n 对应的角度值
12	EXP(n)	求数值 n 的自然指数
13	FLOOR(n)	求小于或等于数值 n 的最大整数
14	GREATEST(n1,n2,n3)	求 n1、n2 和 n3 三个数中最大的一个
15	GREAT (n1,n2)	求 n1、n2 两个数中最大的一个
16	LEAST(n1,n2,n3)	求 n1、n2 和 n3 三个数中最小的一个
17	LN(n)	求数值 n 的自然对数
18	LOG(n1[,n2])	求数值 n2 以 n1 为底数的对数
19	LOG10(n)	求数值 n 以 10 为底的对数
20	MOD(m,n)	求数值 m 被数值 n 除的余数
21	PI()	得到常数 π
22	POWER(n1,n2)/POWER2(n1,n2)	求数值 n2 以 n1 为基数的指数
23	RADIANS(n)	求角度 n 对应的弧度值
24	RAND([n])	求一个 0 到 1 之间的随机浮点数
25	ROUND(n[,m])	求四舍五入值函数
26	SIGN(n)	判断数值的数学符号
27	SIN(n)	求数值 n 的正弦值
28	SINH(n)	求数值 n 的双曲正弦值
29	SQRT(n)	求数值 n 的平方根
30	TAN(n)	求数值 n 的正切值
31	TANH(n)	求数值 n 的双曲正切值
32	TO_NUMBER (char [,fmt])	将 CHAR、VARCHAR、VARCHAR2 等类型的字符串转换为 DECIMAL 类型的数值

33	TRUNC(n[,m])	截取数值函数
34	TRUNCATE(n[,m])	截取数值函数，等价于 TRUNC(n[,m])
35	TO_CHAR(n [, fmt [, 'nls']])	将数值类型的数据转换为 VARCHAR 类型输出
36	BITAND(n1, n2)	求两个数值型数值按位进行 AND 运算的结果

表 8.2 字符串函数

序号	函数名	功能简要说明
01	ASCII(char)	返回字符对应的整数
02	ASCIISTR(char)	将字符串 char 中，非 ASCII 的字符转成\XXXX(UTF-16)格式，ASCII 字符保持不变
03	BIT_LENGTH(char)	求字符串的位长度
04	CHAR(n)	返回整数 n 对应的字符
05	CHAR_LENGTH(char)/ CHARACTER_LENGTH(char)	求字符串的串长度
06	CHR(n)	返回整数 n 对应的字符，等价于 CHAR(n)
07	CONCAT(char1,char2,char3,…)	顺序联结多个字符串成为一个字符串
08	DIFFERENCE(char1,char2)	比较两个字符串的 SOUNDEX 值之差异，返回两个 SOUNDEX 值串同一位置出现相同字符的个数。
09	INITCAP(char)	将字符串中单词的首字符转换成大写的字符
10	INS(char1,begin,n,char2)	删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 串的 begin 所指位置
11	INSERT(char1,n1,n2,char2) / INSSTR(char1,n1,n2,char2)	将字符串 char1 从 n1 的位置开始删除 n2 个字符，并将 char2 插入到 char1 中 n1 的位置
12	INSTR(char1,char2[,n,[m]])	从输入字符串 char1 的第 n 个字符开始查找字符串 char2 的第 m 次出现的位置，以字符计算
13	INSTRB(char1,char2[,n,[m]])	从 char1 的第 n 个字节开始查找字符串 char2 的第 m 次出现的位置，以字节计算
14	LCASE(char)	将大写的字符串转换为小写的字符串
15	LEFT(char,n) / LEFTSTR(char,n)	返回字符串最左边的 n 个字符组成的字符串
16	LEN(char)	返回给定字符串表达式的字符(而不是字节)个数(汉字为一个字符)，其中不包含尾随空格
17	LENGTH(char)	返回给定字符串表达式的字符(而不是字节)个数(汉字为一个字符)，其中包含尾随空格
18	OCTET_LENGTH(char)	返回输入字符串的字节数
19	COPYB(DEST_LOB,SRC_LOB,LEN[,DOFFSET[,SOFFSET]])	拷贝指定长度的源 BLOB 数据插入到目标 BLOB
20	LOCATE(char1,char2[,n])	返回 char1 在 char2 中首次出现的位置
21	LOWER(char)	将大写的字符串转换为小写的字符串
22	LPAD(char1,n,char2)	在输入字符串的左边填充上 char2 指定的字符，将其

		拉伸至 n 个字节长度
23	LTRIM(char1,char2)	从输入字符串中删除所有的前导字符，这些前导字符由 char2 来定义
24	POSITION(char1,/ IN char2)	求串 1 在串 2 中第一次出现的位置
25	REPEAT(char,n) / REPEATSTR(char,n)	返回将字符串重复 n 次形成的字符串
26	REPLACE(STR, search [,replace])	将输入字符串 STR 中所有出现的字符串 search 都替换成字符串 replace ，其中 STR 为 char、clob 或 text 类型
27	REPLICATE(char,times)	把字符串 char 自己复制 times 份
28	REVERSE(char)	将字符串反序
29	RIGHT / RIGHTSTR(char,n)	返回字符串最右边 n 个字符组成的字符串
30	RPAD(char1,n,char2)	类似 LPAD 函数，只是向右拉伸该字符串使之达到 n 个字节长度
31	RTRIM(char1,char2)	从输入字符串的右端开始删除 char2 参数中的字符
32	SOUNDEX(char)	返回一个表示字符串发音的字符串
33	SPACE(n)	返回一个包含 n 个空格的字符串
34	STRPOSDEC(char)	把字符串 char 中最后一个字符的值减一
35	STRPOSDEC(char,pos)	把字符串 char 中指定位置 pos 上的字符值减一
36	STRPOSINC(char)	把字符串 char 中最后一个字符的值加一
37	STRPOSINC(char,pos)	把字符串 char 中指定位置 pos 上的字符值加一
38	STUFF(char1,begin,n,char2)	删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 串的 begin 所指位置
39	SUBSTR(char,m,n) / SUBSTRING(char FROM m [FOR n])	返回 char 中从字符位置 m 开始的 n 个字符
40	SUBSTRB(char,n,m)	SUBSTR 函数等价的单字节形式
41	TO_CHAR(character)	将 VARCHAR、CLOB、TEXT 类型的数据转化为 VARCHAR 类型输出
42	TRANSLATE(char,from,to)	将所有出现在搜索字符集中的字符转换成字符集中的相应字符
43	TRIM([LEADING|TRAILING|BOTH] [exp] [] FROM char2])	删去字符串 char2 中由串 char1 指定的字符
44	UCASE(char)	将小写的字符串转换为大写的字符串
45	UPPER(char)	将小写的字符串转换为大写的字符串
46	REGEXP	根据符合 POSIX 标准的正则表达式进行字符串匹配
47	OVERLAY(char1 PLACING char2 FROM int [FOR int])	字符串覆盖函数,用 char2 覆盖 char1 中指定的子串，返回修改后的 char1
48	TEXT_EQUAL	返回两个 LONGVARCHAR 类型的值的比较结果，相同返回 1，否则返回 0
49	BLOB_EQUAL	返回两个 LONGVARBINARY 类型的值的比较结果，相同返回 1，否则返回 0

50	NLSSORT(str1 [,nls_sort=str2])	返回对汉字排序的编码
51	GREATEST(char1, char2, char3)	求 char 1、char 2 和 char 3 中最大的字符串
52	GREAT (char1, char2)	求 char 1、char 2 中最大的字符串
53	TO_SINGLE_BYTE (char)	将多字节形式的字符（串）转换为对应的单字节形式
54	TO_MULTI_BYTE (char)	将单字节形式的字符（串）转换为对应的多字节形式
55	EMPTY_CLOB ()	初始化 clob 字段
56	EMPTY_BLOB ()	初始化 blob 字段
57	UNISTR (char)	将字符串 char 中，ascii 码（'\XXXX'4 个 16 进制字符格式）转成本地字符。对于其他字符保持不变。
58	ISNULL(char)	判断表达式是否为 NULL

表 8.3 日期时间函数

序号	函数名	功能简要说明
01	ADD_DAYS(date,n)	返回日期加上 n 天后的新日期
02	ADD_MONTHS(date,n)	在输入日期上加上指定的几个月返回一个新日期
03	ADD_WEEKS(date,n)	返回日期加上 n 个星期后的新日期
04	CURDATE()	返回系统当前日期
05	CURTIME(n)	返回系统当前时间
06	CURRENT_DATE()	返回系统当前日期
07	CURRENT_TIME(n)	返回系统当前时间
08	CURRENT_TIMESTAMP(n)	返回系统当前带会话时区信息的时间戳
09	DATEADD(datepart,n,date)	向指定的日期加上一段时间
10	DATEDIFF(datepart,date1,date2)	返回跨两个指定日期的日期和时间边界数
11	DATEPART(datepart,date)	返回代表日期的指定部分的整数
12	DAY(date)	返回日期中的天数
13	DAYNAME(date)	返回日期的星期名称
14	DAYOFMONTH(date)	返回日期为所在月份中的第几天
15	DAYOFWEEK(date)	返回日期为所在星期中的第几天
16	DAYOFYEAR(date)	返回日期为所在年中的第几天
17	DAYS_BETWEEN(date1,date2)	返回两个日期之间的天数
18	EXTRACT(时间字段 FROM date)	抽取日期时间或时间间隔类型中某一个字段的值
19	GETDATE(n)	返回系统当前时间戳
20	GREATEST(date1,date2,date3)	求 date1、date2 和 date3 中的最大日期
21	GREAT (date1,date2)	求 date1、date2 中的最大日期
22	HOUR(time)	返回时间中的小时分量
23	LAST_DAY(date)	返回输入日期所在月份最后一天的日期
24	LEAST(date1, date2, date3)	求 date1、date2 和 date3 中的最小日期

25	MINUTE(time)	返回时间中的分钟分量
26	MONTH(date)	返回日期中的月份分量
27	MONTHNAME(date)	返回日期中月分量的名称
28	MONTHS_BETWEEN(date1,date2)	返回两个日期之间的月份数
29	NEXT_DAY(date1,char2)	返回输入日期指定若干天后的日期
30	NOW(n)	返回系统当前时间戳
31	QUARTER(date)	返回日期在所处年中的季节数
32	SECOND(time)	返回时间中的秒分量
33	ROUND (date1[, fmt])	把日期四舍五入到最接近格式元素指定的形式
34	TIMESTAMPADD(datepart,n,timestamp)	返回时间戳 timestamp 加上 n 个 datepart 指定的时间段的结果
35	TIMESTAMPDIFF(datepart,timeStamp1,timestamp2)	返回一个表明 timestamp2 与 timestamp1 之间的指定 datepart 类型时间间隔的整数
36	SYSDATE()	返回系统的当前日期
37	TO_DATE(CHAR[,fmt[,'nls']]) /TO_TIMESTAMP(CHAR[,fmt[,'nls']]) / TO_TIMESTAMP_TZ(CHAR[,fmt])	字符串转换为日期时间数据类型
38	FROM_TZ(timestamp,timezone|tz_name])	将时间戳类型 timestamp 和时区类型 timezone（或时区名称 tz_name）转化为 timestamp with timezone 类型
39	TRUNC(date[,fmt])	把日期截断到最接近格式元素指定的形式
40	WEEK(date)	返回日期为所在年中的第几周
41	WEEKDAY(date)	返回当前日期的星期值
42	WEEKS_BETWEEN(date1,date2)	返回两个日期之间相差周数
43	YEAR(date)	返回日期的年分量
44	YEARS_BETWEEN(date1,date2)	返回两个日期之间相差年数
45	LOCALTIME(n)	返回系统当前时间
46	LOCALTIMESTAMP(n)	返回系统当前时间戳
47	OVERLAPS	返回两个时间段是否存在重叠
48	TO_CHAR(date[,fmt[,nls]])	将日期数据类型 DATE 转换为一个在日期语法 fmt 中指定语法的 VARCHAR 类型字符串。
49	SYSTIMESTAMP(n)	返回系统当前带数据库时区信息的时间戳
50	NUMTODSINTERVAL(dec,interval_unit)	转换一个指定的 DEC 类型到 INTERVAL DAY TO SECOND
51	NUMTOYMINTERVAL (dec,interval_unit)	转换一个指定的 DEC 类型值到 INTERVAL YEAR TO MONTH

52	WEEK(date, mode)	根据指定的 mode 计算日期为年中的第几周
53	UNIX_TIMESTAMP (datetime)	返回自标准时区的'1970-01-01 00:00:00 +0:00'的到本地会话时区的指定时间的秒数差
54	FROM_UNIXTIME(unixtime)	返回将自'1970-01-01 00:00:00'的秒数差转成本地会话时区的时间戳类型
	FROM_UNIXTIME(unixtime, fmt)	将自'1970-01-01 00:00:00'的秒数差转成本地会话时区的指定 fmt 格式的时间串
55	SESSIONTIMEZONE	返回当前会话的时区
56	DATE_FORMAT(d, format)	以不同的格式显示日期/时间数据
57	TIME_TO_SEC(d)	将时间换算成秒
58	SEC_TO_TIME(sec)	将秒换算成时间
59	TO_DAYS(timestamp)	转换成公元 0 年 1 月 1 日的天数差

表 8.4 空值判断函数

序号	函数名	功能简要说明
01	COALESCE(n1,n2,…nx)	返回第一个非空的值
02	IFNULL(n1,n2)	当 n1 为非空时，返回 n1；若 n1 为空，则返回 n2
03	ISNULL(n1,n2)	当 n1 为非空时，返回 n1；若 n1 为空，则返回 n2
04	NULLIF(n1,n2)	如果 n1=n2 返回 NULL，否则返回 n1
05	NVL(n1,n2)	返回第一个非空的值
06	NULL_EQU	返回两个类型相同的值的比较

表 8.5 类型转换函数

序号	函数名	功能简要说明
01	CAST(value AS 类型说明)	将 value 转换为指定的类型
02	CONVERT(类型说明,value)	将 value 转换为指定的类型
03	HEXTORAW(exp)	将 exp 转换为 BLOB 类型
04	RAWTOHEX(exp)	将 exp 转换为 VARCHAR 类型
05	BINTOCHAR(exp)	将 exp 转换为 CHAR
06	TO_BLOB(value)	将 value 转换为 blob

表 8.6 杂类函数

序号	函数名	功能简要说明
01	DECODE(exp, search1, result1, … searchn, resultn [,default])	查表译码
02	ISDATE(exp)	判断表达式是否为有效的日期
03	ISNUMERIC(exp)	判断表达式是否为有效的数值
04	DM_HASH (exp)	根据给定表达式生成 HASH 值
05	LNNVL(condition)	根据表达式计算结果返回布尔值
06	LENGTHB(value)	返回 value 的字节数
07	FIELD(value, e1, e2, e3, e4...en)	返回 value 在列表 e1, e2, e3, e4...en 中的位置序号，不在输入列表时则返回 0

8.1 数值函数

数值函数接受数值参数并返回数值作为结果。

1. 函数 ABS

```
语法：ABS(n)
```

功能：返回 n 的绝对值。n 必须是数值类型。

例 查询现价小于 10 元或大于 20 元的信息。

```
SELECT  PRODUCTID,NAME  FROM  PRODUCTION.PRODUCT
WHERE  ABS(NOWPRICE-15)>5;
```

查询结果如下表 8.1.1 所示。

表 8.1.1

PRODUCTID	NAME
3	老人与海
4	射雕英雄传(全四册)
6	长征
7	数据结构(C 语言版)(附光盘)
10	噼里啪啦丛书(全 7 册)

2. 函数 ACOS

```
语法：ACOS(n)
```

功能：返回 n 的反余弦值。n 必须是数值类型，且取值在-1 到 1 之间，函数结果从 0 到 π。

例

```
SELECT ACOS(0);
```

查询结果为：1.570796326794897E+000

3. 函数 ASIN

```
语法：ASIN(n)
```

功能：返回 n 的反正弦值。 n 必须是数值类型， 且取值在-1 到 1 之间， 函数结果从 -π/2 到 π/2。

例

```
SELECT ASIN(0);
```

查询结果为：0.000000000000000E+000

4. 函数 ATAN

```
语法：ATAN(n)
```

功能：返回 n 的反正切值。n 必须是数值类型，取值可以是任意大小，函数结果从-π/2 到 π/2。

例

```
SELECT ATAN(1);
```

查询结果为：7.853981633974483E-001

5. 函数 ATAN2

```
语法：ATAN2(n, m)
```

功能：返回 n/m 的反正切值。n,m 必须是数值类型，取值可以是任意大小，函数结果从-π/2 到π/2。

例

```
SELECT ATAN2(0.2,0.3);
```

查询结果为：5.880026035475676E-001

6. 函数 CEIL

```
语法：CEIL(n)
```

功能：返回大于等于 n 的最小整数。n 必须是数值类型。返回类型与 n 的类型相同。

例

```
SELECT  CEIL(15.6);
```

查询结果为：16

```
SELECT  CEIL(-16.23);
```

查询结果为：-16

7. 函数 CEILING

```
语法：CEILING(n)
```

功能：返回大于等于 n 的最小整数。等价于函数 CEIL(n)。

8. 函数 COS

```
语法：COS(n)
```

功能：返回 n 的余弦值。n 必须是数值类型，是用弧度表示的值。将角度乘以π/180，可以转换为弧度值。

例

```
SELECT COS(14.78);
```

查询结果为：-5.994654261946543E-001

9. 函数 COSH

```
语法：COSH(n)
```

功能：返回 n 的双曲余弦值。

例

```
SELECT COSH(0)"Hyperbolic cosine of 0";
```

查询结果为：1.000000000000000E+000

10. 函数 COT

```
语法：COT(n)
```

功能：返回 n 的余切值。n 必须是数值类型，是用弧度表示的值。将角度乘以π/180，可以转换为弧度值。

例

```
SELECT COT(20 * 3.1415926/180);
```

查询结果为：2.747477470356783E+000

11. 函数 DEGREES

```
语法：DEGREES(n)
```

功能：返回弧度 n 对应的角度值，返回值类型与 n 的类型相同。

例

```
SELECT DEGREES(1.0);
```

查询结果为：5.729577951308238E+001

12. 函数 EXP

```
语法：EXP(n)
```

功能：返回 e 的 n 次幂。

例

```
SELECT EXP(4) "e to the 4th power";
```

查询结果为：5.459815003314424E+001

13. 函数 FLOOR

```
语法：FLOOR(n)
```

功能：返回小于等于 n 的最大整数值。n 必须是数值类型。返回类型与 n 的类型相同。

例

```
SELECT  FLOOR(15.6);
```

查询结果为：15.0

```
SELECT  FLOOR(-16.23);
```

查询结果为：-17.0

14. 函数 GREATEST

```
语法：GREATEST(n1,n2,n3)
```

功能：求 n1、n2 和 n3 中最大的数。

例

```
SELECT GREATEST(1.2,3.4,2.1);
```

查询结果：3.4

15. 函数 GREAT

```
语法：GREAT(n1,n2)
```

功能：求 n1、n2 中的最大的数。

例

```
SELECT GREAT (2, 4);
```

查询结果：4

16. 函数 LEAST

```
语法：LEAST(n1,n2,n3)
```

功能：求n1、n2和n3三个数中最小的一个。

例

```
SELECT LEAST(1.2,3.4,2.1);
```

查询结果：1.2

17. 函数 LN

```
语法：LN(n)
```

功能：返回 n 的自然对数。n 为数值类型，且大于 0。

例

```
SELECT LN(95) "Natural log of 95";
```

查询结果为：4.553876891600541E+000

18．函数 LOG

```
语法：LOG(m[,n])
```

功能：返回数值 n 以数值 m 为底的对数；若参数 m 省略，返回 n 的自然对数。m,n 为数值类型，m 大于 0 且不为 1。

例

```
SELECT LOG(10,100);
```

查询结果为：2.000000000000000E+000

```
SELECT LOG(95);
```

查询结果为：4.553876891600541E+000

19．函数 LOG10

```
语法：LOG10(n)
```

功能：返回数值 n 以 10 为底的对数。n 为数值类型，且大于 0。

例

```
SELECT LOG10(100);
```

查询结果为：2.000000000000000E+000

20．函数 MOD

```
语法：MOD(m,n)
```

功能：返回 m 除以 n 的余数，当 n 为 0 时直接返回 m。m,n 为数值类型。

例

```
SELECT ROUND(NOWPRICE),mod(ROUND(NOWPRICE),10) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 8.1.2 所示。

表 8.1.2

ROUND(NOWPRICE)	mod(ROUND(NOWPRICE),10)
15	5
14	4
6	6
22	2
20	0
38	8
26	6
11	1
11	1
42	2

21．函数 PI

```
语法：PI()
```

功能：返回常数 π。

例

```
SELECT PI();
```

查询结果为：3.141592653589793E+000

22．函数 POWER/POWER2

```
语法：POWER(m,n) / POWER2(m,n)
```

功能：返回 m 的 n 次幂。m,n 为数值类型，如果 m 为负数的话，n 必须为一个整数。其中 POWER()的返回值类型为 DOUBLE，POWER2()的返回值类型为 DECIMAL。

例

```
SELECT POWER(3,2) "Raised";
```

查询结果为：9.000000000000000E+000

```
SELECT POWER(-3,3) "Raised";
```

查询结果为：-2.700000000000000E+001

23．函数 RADIANS()

```
语法：RADIANS(n)
```

功能：返回角度 n 对应的弧度值，返回值类型与 n 的类型相同。

例

```
SELECT RADIANS(180.0);
```

查询结果为：3.141592653589790E+000

24．函数 RAND()

```
语法：RAND([n])
```

功能：返回一个[0,1]之间的随机浮点数。n 为数值类型，为生成随机数的种子，当 n 省略时，系统自动生成随机数种子。

例

```
SELECT RAND();
```

查询结果为一个随机生成的小数

```
SELECT RAND(314);
```

查询结果为：3.247169408246101E-002

25．函数 ROUND

```
语法：ROUND(n [,m])
```

功能：返回四舍五入到小数点后面 m 位的 n 值。m 应为一个整数，缺省值为 0，m 为负整数则四舍五入到小数点的左边，m 为正整数则四舍五入到小数点的右边。若 m 为小数，系统将自动将其转换为整数。

例

```
SELECT NOWPRICE,ROUND(NOWPRICE) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 8.1.3 所示。

表 8.1.3

NOWPRICE	ROUND(NOWPRICE)
15.2000	15
14.3000	14
6.1000	6
21.7000	22

20.0000	20
37.7000	38
25.5000	26
11.4000	11
11.1000	11
42.0000	42

```
SELECT  ROUND(15.163,-1);
```

查询结果为：20.0

```
SELECT  ROUND(15.163);
```

查询结果为：15

26. 函数 SIGN

语法：SIGN(n)

功能：如果 n 为正数，SIGN(n)返回 1，如果 n 为负数，SIGN(n)返回-1，如果 n 为 0，SIGN(n)返回 0。

例

```
SELECT ROUND(NOWPRICE),SIGN(ROUND(NOWPRICE)-20) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 8.1.4 所示。

表 8.1.4

ROUND(NOWPRICE)	SIGN(ROUND(NOWPRICE)-20)
15	-1
14	-1
6	-1
22	1
20	0
38	1
26	1
11	-1
11	-1
42	1

27. 函数 SIN

语法：SIN(n)

功能：返回 n 的正弦值。n 必须是数值类型，是用弧度表示的值。将角度乘以 π/180，可以转换为弧度值。

例

```
SELECT SIN(0);
```

查询结果为：0.0

28. 函数 SINH

语法：SINH(n)

功能：返回 n 的双曲正弦值。

例

```
SELECT SINH(1);
```

查询结果为：1.175201193643801E+000

29. 函数 SQRT

```
语法：SQRT(n)
```

功能：返回 n 的平方根。n 为数值类型，且大于等于 0。

例

```
SELECT ROUND(NOWPRICE),SQRT (ROUND(NOWPRICE)) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 8.1.5 所示。

表 8.1.5

ROUND(NOWPRICE)	SQRT(ROUND(NOWPRICE))
15	3.872983346207417E+000
14	3.741657386773941E+000
6	2.449489742783178E+000
22	4.690415759823430E+000
20	4.472135954999580E+000
38	6.164414002968976E+000
26	5.099019513592785E+000
11	3.316624790355400E+000
11	3.316624790355400E+000
42	6.480740698407860E+000

30. 函数 TAN

```
语法：TAN(n)
```

功能：返回 n 的正切值。n 必须是数值类型，是用弧度表示的值。将角度乘以 π/180，可以转换为弧度值。

例

```
SELECT TAN(45*Pi()/180);
```

查询结果为：9.999999999999999E-001

31. 函数 TANH

```
语法：TANH(n)
```

功能：返回 n 的双曲正切值。

例

```
SELECT TANH(0);
```

查询结果为：0.000000000000000E+000

32. 函数 TO_NUMBER

```
语法：TO_NUMBER (char [,fmt])
```

功能：将 CHAR、VARCHAR、VARCHAR2 等类型的字符串转换为 DECIMAL 类型的数值。char 为待转换的字符串，fmt 为目标格式串。

若指定了 fmt 格式则转换后的 char 应该遵循相应的数字格式，若没有指定则直接转换成 DECIMAL。fmt 格式串一定要包容实际字符串的数据的格式，否则报错。无格式转换中只支持小数点和正负号。合法的 fmt 格式串字符如下表：

表 8.1.6

元素	例子	说明
,(逗号)	9,999	指定位置处返回逗号 注意：1.逗号不能开头 2.不能在小数点右边
.(小数点)	99.99	指定位置处返回小数点
$	$9999	美元符号开头
0	0999 9990	以 0 开头，返回指定字符的数字 以 0 结尾，返回指定字符的数字
9	9999	返回指定字符的数字，如果不够正号以空格代替， 负号以-代替，0 开头也以空格代替。
D	99D99	返回小数点的指定位置，默认为'.',格式串中最多能有一个 D
G	9G999	返回指定位置处的组分隔符，可有多个，但不能出现在小数点右边
S	S9999 9999S	负值前面返回一个-号 正值前面不返回任何值 负值后面返回一个-号 正值后面不返回任何值 只能在格式串首尾出现
X	XXXX xxxx	返回指定字符的十六进制值，如果不是整数则四舍五入到整数， 如果为负数则返回错误。
C	C9999	返回指定字符的数字
B	B9999	返回指定字符的数字

例 1 使用 9、G、D 来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('2,222.22', '9G999D99');
```

查询结果：2222.22

例 2 使用 9、,(逗号)、.(小数点)来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('2,222.22', '9,999.99');
```

查询结果：2222.22

例 3 使用$、9、,(逗号)、.(小数点)来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('$2,222.22','$9G999D99');
```

查询结果：2222.22

例 4 使用 S、9 和.(小数点)来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('-1212.12','S9999.99');
```

查询结果：-1212.12

例 5 使用 xxxx 来转换字符串'1,234'。

```
SELECT TO_NUMBER('1,234','XXXX');
```

查询结果：4660

例 6 无格式转换。

```
SELECT TO_NUMBER('-123.4');
```

查询结果：-123.4

33.函数 TRUNC

```
语法：TRUNC(n [,m])
```

功能：将数值 n 的小数点后第 m 位以后的数全部截去。当数值参数 m 为负数时表示将数值 n 的小数点前的第 m 位截去。当数值参数 m 省略时，m 默认为 0。

特殊说明：当 m 为负数，其绝对值大于或等于 n 的整数位个数时，结果取 0；当 m 取正数，其值大于等于 n 的小数位个数时，结果取 n。

例

```
SELECT SQRT(NOWPRICE), TRUNC(SQRT (ROUND(NOWPRICE) ),1)FROM PRODUCTION.PRODUCT;
```

查询结果如下表 8.1.7 所示。

表 8.1.7

SQRT (NOWPRICE)	TRUNC(SQRT (ROUND(NOWPRICE)),1)
3.8987177379235853	3.8
3.7815340802378077	3.7
2.4698178070456938	2.4
4.658325879540846	4.6
4.47213595499958	4.4
6.1400325732035	6.1
5.049752469181039	5.0
3.3763886032268267	3.3
3.331666249791536	3.3
6.48074069840786	6.4

```
SELECT  TRUNC(15.167,-1);
```

查询结果为：10.000

34．函数 TRUNCATE

```
语法：TRUNCATE(n  [,m])
```

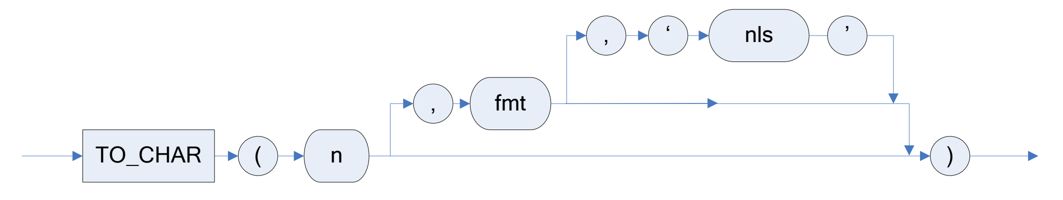
功能：等价于 TRUNC(n [,m])。将数值 n 的小数点后第 m 位以后的数全部截去。当数值参数 m 为负数时表示将数值 n 的小数点前的第 m 位截去。当数值参数 m 省略时，m 默认为 0。

特殊说明：当 m 为负数，其绝对值大于或等于 n 的整数位个数时，结果取 0；当 m 取正数，其值大于等于 n 的小数位个数时，结果取 n。

35．函数 TO_CHAR

```
语法：TO_CHAR(n [, fmt [, 'nls' ] ])
```

图例



语句功能：

将数值类型的数据转化为 VARCHAR 类型输出。其中：n 为数值类型的数据；fmt 为目标格式串。DM 的缺省格式为数字的字符串本身。如 SELECT TO_CHAR(11.18)，查询结果为：11.18。

fmt 中包含的格式控制符主要可以分为三类，具体如下如下：

1） 主体标记；
2） 前缀标记；
3） 后缀标记。

其中主体标记包含的标记如表 8.1.8 所示。

表 8.1.8 主体标记

格式控制符	说明

逗号（,）	逗号只能出现在整数部分的任意位置，如 to_char(1234, '9,99,9')，结果为 1,23,4
点号（.）	作为小数点分隔符，不足的位数由后面的掩码决定
0	表示位数不足的时候用0填充，如to_char(1234, '09999.00')，结果为01234.00
9	表示位数不足的时候用空格填充，如to_char(1234, '9999,99')，结果为’ 12,34’
D	作用同点号
G	作用同逗号
X	表示16进制
V	表示10的n次方
RN	转换为大写的罗马数字
rn	转换为小写的罗马数字

其中前缀标记包含的标记如表 8.1.9 所示。

表 8.1.9 前缀标记

格式控制符	说明
FM	去掉前置空格
$	美元符号。只能放在掩码最前面，且只能有一个
B	当整数部分的值为零时，返回空格
S	表示正负号，如 to_char(1234, 'S9999') 结果为 +1234，to_char(-1234,'S9999') 结果为-1234
TM9	64个字符内返回原数值，超过则返回科学计数值
TME	返回科学计数值
C	当前货币名称缩写
L	当前货币符号

其中后缀标记包含的标记如表 8.1.10 所示。

表 8.1.10 后缀标记

格式控制符	说明
EEEE	科学计数符
MI	如'9999MI'，如果是负数，在尾部加上负号(-)；如果是正数，则尾部加上空格
PR	将负数放到尖括号<>中
C	当前货币名称缩写
L	当前货币符号
S	表示正负号

这些标记的组合规则主要包括以下几个：

1） 前缀之间的冲突；

2） 后缀与前缀之间的冲突；

3） 后缀之间的冲突。

其中，前缀之间的冲突如表 8.1.11 所示。

表 8.1.11 前缀之间的冲突

前缀	与指定前缀存在冲突的前缀

$	$, C, L
B	B
S	$, B, S, C, L
TM9	$, B, S
TME	$, B, S
FM	$, B, TM9, TME
C	C, L,$
L	C, L,$

注：前缀之间的冲突指上表中第二列的前缀不能放在第一列的前缀之前。

如当前缀为 S 时，前缀中不能还有$, B, S, C, L 标记，即$S，BS，SS，CS，LS 不能作为前缀。类似，对于前缀 L，则 CL，LL，$L 不能作为前缀。

后缀与前缀之间的冲突如表 8.1.12 所示。

表 8.1.12 后缀与前缀之间的冲突

后缀	与指定后缀存在冲突的前缀
L	L, C, $
C	L, C, $
$	$, C, L, MI, PR
S	S
PR	S
MI	S

如当后缀为 C 时，前缀中不能还有 L, C, $等标记，如格式'L999C'等。

后缀之间的冲突如表 8.1.13 所示。

表 8.1.13 后缀之间的冲突

后缀	与指定后缀存在冲突的后缀
EEEE	S, EEEE, MI, PR
S	S, MI, PR
PR	S, MI, PR
MI	S, MI, PR
C	C, L, MI, PR, S, EEEE, $
L	C, L, MI, PR, S, EEEE, $
$	$, MI, PR, S, C, L

注：后缀之间的冲突指上表中第二列的后缀不能放在第一列的后缀之前。

如当后缀为 L 时，后缀中不能还有 C、L、MI、PR、S、EEEE、￥等标记，即后缀 CL，LL，MIL，PRL，SL，EEEEL 不能在格式字符串中出现。

nls 用来指定以下数字格式元素返回的字符：

1） 小数点字符；

2） 组分隔符；

3） 本地货币符号。

4） 国际货币符号。

这个参数可以有这种形式：

```
'NLS_NUMERIC_CHARACTERS = ''dg''
NLS_CURRENCY = ''text''
NLS_ISO_CURRENCY = territory '
```

其中：NLS_NUMERIC_CHARACTERS 参数指定字符 D 和 G 代表小数点字符和组分隔，必须用引号引起来。NLS_NUMERIC_CHARACTERS 串的长度只能是两个，并且这两个字符的不能相同；NLS_CURRENCY 指定的字符串用来代替本地货币符号，仅当 FMT 的前缀中有 L 时有效，不能超过 10 个字符的长度。nls 参数字符串如果包含空格，要用单引号括起来；如果包含单引号，也要用单引号括起来，以对单引号进行转义。

NLS_ISO_CURRENCY 用来指定的字符串用来代替国际货币符号，仅当 FMT 的前缀中有 C 时有效，取值只能是表 8.1.14 中的值，得到的结果是缩写的内容。

表 8.1.14 NLS_ISO_CURRENCY 的值及缩写形式

NLS_TERRITORY	**缩写**
CHINA	CNY
TAIWAN	TWD
AMERICA	USD
UNITED KINGDOM	GBP
CANADA	CAD
FRANCE	EUR
GERMANY	EUR
ITALY	EUR
JAPAN	JPY
KOREA	KRW
BRAZIL	BRL
PORTUGAL	EUR

举例说明

```
SELECT TO_CHAR('01110' + 1);
```

查询结果为：1111

```
SELECT TO_CHAR(-10000,'L99G999D99MI') "Amount";
```

查询结果为：

Amount

￥10,000.00-

```
CREATE TABLE T_INT (C1 INT);
INSERT INTO T_INT VALUES(456),(0),(213);
SELECT TO_CHAR(C1, 'L999D99MI',
  'NLS_NUMERIC_CHARACTERS = '';:''
  NLS_CURRENCY = ''AusDollars'' ') FROM T_INT;
```

查询结果为：

AusDollars456;00

AusDollars;00

AusDollars213;00

```
SELECT TO_CHAR(C1, 'C999D99MI',
'NLS_NUMERIC_CHARACTERS = '';:''
NLS_CURRENCY = ''AusDollars''
```

```
NLS_ISO_CURRENCY = ''TAIWAN'' ') FROM T_INT;
```

查询结果为：

TWD456;00

TWD;00

TWD213;00

36. 函数 BITAND

```
语法：BITAND(n1,  n2)
```

功能：返回两个数值型数值 n1 和 n2 按位进行 AND 运算后的结果。

特殊说明：当 n1 或 n2 是小数时，去掉小数点后做 AND 运算；如果 n1 或 n2 有一个是 0，则结果是 0；如果 n1 或 n2 有一个是 null，则结果是 null。

例

```
SELECT BITAND(-4, -5);
```

查询结果：-8

8.2 字符串函数

字符串函数一般接受字符类型(包括 CHAR 和 VARCHAR)和数值类型的参数，返回值一般是字符类型或是数值类型。

1． 函数 ASCII

```
语法：ASCII(char)
```

功能：返回字符 char 对应的整数（ASSIC 值）。

例

```
SELECT  ASCII('B') ,ASCII('中');
```

查询结果为：66 54992

2． 函数 ASCIISTR

```
语法：ASCIISTR (char)
```

功能：将字符串 char 中，非 ASCII 的字符转成\XXXX(UTF-16)格式，ASCII 字符保持不变。

例 非 unicode 库下，执行如下操作：

```
SELECT CHr(54992),ASCIISTR('中') ,ASCIISTR(CHr(54992));
```

查询结果为：中 \4E2D \4E2D

3． 函数 BIT_LENGTH

```
语法：BIT_LENGTH(char)
```

功能：返回字符串的位(bit)长度。

例

```
SELECT BIT_LENGTH('ab');
```

查询结果为：16

4． 函数 CHAR

```
语法：CHAR(n)
```

功能：返回整数 n 对应的字符。

例

```
SELECT  CHAR(66),CHAR(67),CHAR(68) , CHAR(54992);
```

查询结果为：B C D 中

5. 函数 CHAR_LENGTH / CHARACTER_LENGTH

```
语法：CHAR_LENGTH(char) 或 CHARACTER_LENGTH(char)
```

功能：返回字符串 char 的长度，以字符作为计算单位，一个汉字作为一个字符计算。字符串尾部的空格也计数。

例

```
SELECT  NAME,CHAR_LENGTH(TRIM(BOTH ' ' FROM NAME))
FROM PRODUCTION.PRODUCT;
```

查询结果如下表 8.2.1 所示。

表 8.2.1

NAME	CHAR_LENGTH(TRIM(BOTH ' ' FROM NAME))
红楼梦	3
水浒传	3
老人与海	4
射雕英雄传(全四册)	10
鲁迅文集(小说、散文、杂文)全两册	17
长征	2
数据结构(C 语言版)(附光盘)	15
工作中无小事	6
突破英文基础词汇	8
噼里啪啦丛书(全 7 册)	11

```
SELECT CHAR_LENGTH('我们');
```

查询结果为：2

6. 函数 CHR

```
语法：CHR(n)
```

功能：返回整数 n 对应的字符。等价于 CHAR(n)。

7. 函数 CONCAT

```
语法：CONCAT(char1,char2,char3…)
```

功能：返回多个字符串顺序联结成的一个字符串，该函数等价于连接符||。

例

```
SELECT  PRODUCTID,NAME, PUBLISHER, CONCAT(PRODUCTID,NAME,PUBLISHER)  FROM
PRODUCTION.PRODUCT;
```

查询结果如下表 8.2.2 所示：

表 8.2.2

PRODUCTID	NAME	PUBLISHER	CONCAT(PRODUCTID,NAME,PUBLISHER)
1	红楼梦	中华书局	1 红楼梦中华书局
2	水浒传	中华书局	2 水浒传中华书局

3	老人与海	上海出版社	3 老人与海上海出版社
4	射雕英雄传(全四册)	广州出版社	4 射雕英雄传(全四册)广州出版社
5	鲁迅文集(小说、散文、杂文)全两册		5 鲁迅文集(小说、散文、杂文)全两册
6	长征	人民文学出版社	6 长征人民出版社
7	数据结构(C 语言版)(附光盘)	清华大学出版社	7 数据结构(C 语言版)(附光盘)清华大学出版社
8	工作中无小事	机械工业出版社	8 工作中无小事机械工业出版社
9	突破英文基础词汇	外语教学与研究出版社	9 突破英文基础词汇外语教学与研究出版社
10	噼里啪啦丛书(全 7 册)	21 世纪出版社	10 噼里啪啦丛书(全 7 册)21 世纪出版社

8. 函数 DIFFERENCE()

```
语法：DIFFERENCE(char1,char2)
```

功能：比较两个字符串的 SOUNDEX 值之间的差异，返回两个 SOUNDEX 值串同一位置出现相同字符的个数。

例

```
SELECT  DIFFERENCE('she', 'he');
```

查询结果为：3

9. 函数 INITCAP

```
语法：INITCAP(char)
```

功能：返回句子字符串中，每一个单词的第一个字母改为大写，其他字母改为小写。单词用空格分隔，不是字母的字符不受影响。

例

```
SELECT   INITCAP('hello world');
```

查询结果为：Hello World

10. 函数 INS

```
语法： INS(char1,begin,n,char2)
```

功能：删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 串的 begin 所指位置。

例

```
SELECT  INS ('abcdefg',1,3, 'kkk') ;
```

查询结果为：kkkdefg

11. 函数 INSERT / INSSTR

```
语法： INSERT(char1,n1,n2,char2) / INSSTR(char1,n1,n2,char2)
```

功能：将字符串 char1 从 n1 的位置开始删除 n2 个字符，并将 char2 插入到 char1 中 n1 的位置。

例

```
SELECT  INSERT('That is a cake',2,3, 'his') ;
```

查询结果为：This is a cake

12. 函数 INSTR

```
语法：INSTR(char1,char2[,n[,m]])
```

功能：返回 char1 中包含 char2 的特定位置。INSTR 从 char1 的左边开始搜索，开始位置是 n，如果 n 为负数，则搜索从 char1 的最右边开始，当搜索到 char2 的第 m 次出现时，返回所在位置。n 和 m 的缺省值都为 1，即返回 char1 中第一次出现 char2 的位置，这时与 POSITION 相类似。如果从 n 开始没有找到第 m 次出现的 char2，则返回 0。n 和 m 以字符作为计算单位，一个西文字符和一个汉字都作为一个字符计算。

此函数中 char1 和 char2 可以是 CHAR 或 VARCHAR 数据类型，n 和 m 是数值类型。

例

```
SELECT INSTR('CORPORATE FLOOR', 'OR', 3, 2) "Instring";
```

查询结果为：14

```
SELECT INSTR('我们的计算机', '计算机',1,1);
```

查询结果为：4

13. 函数 INSTRB

```
语法：INSTRB(char1,char2[,n[,m]])
```

功能：返回从 char1 的第 n 个字节开始查找字符串 char2 的第 m 次出现的位置。INSTRB 从 char1 的左边开始搜索，开始位置是 n，如果 n 为负数，则搜索从 char1 的最右边开始，当搜索到 char2 的第 m 次出现时，返回所在位置。n 和 m 的缺省值都为 1，即返回 char1 中第一次出现 char2 的位置，这时与 POSITION 相类似。如果从 n 开始没有找到第 m 次出现的 char2，则返回 0。以字节作为计算单位，一个汉字根据编码类型不同可能占据 2 个或多个字节。

此函数中 char1 和 char2 可以是 CHAR 或 VARCHAR 数据类型，n 和 m 是数值类型。

例

```
SELECT INSTRB('CORPORATE FLOOR', 'OR', 3, 2) "Instring";
```

查询结果为：14

```
SELECT INSTRB('我们的计算机', '计算机',1,1);
```

查询结果为：7

14. 函数 LCASE

```
语法：LCASE(char)
```

功能：返回字符串中，所有字母改为小写，不是字母的字符不受影响。

例

```
SELECT LCASE('ABC');
```

查询结果为：abc

15. 函数 LEFT / LEFTSTR

```
语法：LEFT(char,n) / LEFTSTR(char,n)
```

功能：返回字符串最左边的 n 个字符组成的字符串。

例

```
SELECT  NAME,LEFT(NAME,2)  FROM  PRODUCTION.PRODUCT;
```

查询结果如下表 8.2.3 所示。

表 8.2.3

NAME	LEFT(NAME,2)
红楼梦	红楼

水浒传	水浒
老人与海	老人
射雕英雄传(全四册)	射雕
鲁迅文集(小说、散文、杂文)全两册	鲁迅
长征	长征
数据结构(C 语言版)(附光盘)	数据
工作中无小事	工作
突破英文基础词汇	突破
噼里啪啦丛书(全 7 册)	噼里

```
SELECT  LEFT ('computer science',10);
```

查询结果为：computer s

16. 函数 LEN

```
语法：LEN(char)
```

功能：返回给定字符串表达式的字符(而不是字节)个数，其中不包含尾随空格。

例

```
SELECT LEN ('hi,你好□□');
```

查询结果为：5

说明：□表示空格字符

17. 函数 LENGTH

```
语法：LENGTH(char)
```

功能：返回给定字符串表达式的字符(而不是字节)个数，其中包含尾随空格。

例

```
SELECT LENGTH('hi,你好□□');
```

查询结果为：7

说明：□表示空格字符

18. 函数 OCTET_LENGTH

```
语法：OCTET_LENGTH(char)
```

功能：返回字符串 char 的长度，以字节作为计算单位，一个汉字根据编码类型不同可能占据 2 个或多个字节。

例

```
SELECT OCTET_LENGTH('大家好') "Length in bytes";
```

查询结果为：6

19. 函数 COPYB

```
语法：  COPYB(
         DEST_LOB IN OUT BLOB,
         SRC_LOB IN BLOB,
         LEN IN INTEGER,
         DOFFSET IN INTEGER,
         SOFFSET IN INTEGER
```

```
);
```

功能：拷贝指定长度的源 BLOB 数据插入到目标 BLOB

参数：

dest_lob：目标大对象

src_lob：源大对象

amount：需要拷贝的字节数

doffset：写入的起始偏移

soffset：读取的起始偏移

例

```
CREATE OR REPLACE PROCEDURE PCOPYB1 is
blb1 BLOB;
begin
    copyb(blb1,'abcd123',5);
   SELECT blb1;
end;
/
CALL PCOPYB1;
```

查询结果：0x0ABCD123

```
CREATE OR REPLACE PROCEDURE PCOPYB2 is
blb1 BLOB;
begin
         blb1 = 'abcdeabcdeabcde';
    copyb(blb1,'5465421871321564312354',5,3);
    SELECT blb1;
end;
/
CALL PCOPYB2;
```

查询结果：0x0ABC546542

```
CREATE OR REPLACE PROCEDURE PCOPYB3 is
blb1 BLOB;
begin
         blb1 = 'abcdeabcdeabcde ';
    copyb(blb1,'5465421871321564312354',5,3,5);
    SELECT blb1;
end;
/
CALL PCOPYB3;
```

查询结果：0x0ABC713215

20．函数 LOCATE

```
语法：LOCATE(char1,char2[,n])
```

功能：返回字符串 char1 在 char2 中从位置 n 开始首次出现的位置，如果参数 n 省略或为负数，则从 char2 的最左边开始找。

例

```
SELECT LOCATE('man', 'The manager is a man', 10);
```

查询结果为：18

```
SELECT LOCATE('man', 'The manager is a man');
```

查询结果为：5

21．函数 LOWER

```
语法：LOWER(char)
```

功能：将字符串中的所有大写字母改为小写，其他字符不变。等价于 LCASE(char)。

22．函数 LPAD

```
语法：LPAD(char1,length[,char2])
```

功能：在字符串 char1 的左边，依次加入 char2 中的字符，直到总长度达到 length，返回增加后的字符串。如果未指定 char2，缺省值为空格。length 为正整数。如果 length 的长度比 char1 大，则返回 char2 的前(length-length(char1))个字符+char1，总长度为 length。如果 length 比 char1 小，则返回 char1 的前 lengh 个字符。长度以字节作为计算单位，一个汉字作为二个字节计算。

注：若 length 为小于或等于零的整数，则返回 NULL。

例

```
SELECT LPAD(LPAD('FX',19,'Teacher'),22,'BIG') "LPAD example";
```

查询结果为：BIGTeacherTeacherTeaFX

```
SELECT LPAD('计算机',8, '我们的');
```

查询结果为：我计算机

23．函数 LTRIM

```
语法：LTRIM(char[,set])
```

功能：删除字符串 char 左边起，出现在 set 中的任何字符，当遇到不在 set 中的第一个字符时结果被返回。set 缺省为空格。

例

```
SELECT LTRIM('xyyxxxXxyLAST WORD', 'xy') "LTRIM example";
```

查询结果为：XxyLAST WORD

```
SELECT LTRIM('我们的计算机', '我们');
```

查询结果为：的计算机

24．函数 POSITION

```
语法：POSITION(char1 IN char2) / POSITION(char1, char2)
```

功能：返回在 char2 串中第一次出现的 char1 的位置，如果 char1 是一个零长度的字符串，POSITION 返回 1，如果 char2 中 char1 没有出现，则返回 0。以字节作为计算单位，一个汉字根据编码类型不同可能占据 2 个或多个字节。

例

```
SELECT  POSITION('数' IN '达梦数据库');
```

查询结果为：5

25．函数 REPEAT / REPEATSTR

```
语法：REPEAT(char,n) / REPEATSTR(char,n)
```

功能：返回将字符串重复 n 次形成的字符串。

例

```
SELECT REPEAT ('Hello ',3);
```

查询结果为：Hello Hello Hello

26. 函数 REPLACE

```
语法：REPLACE(STR, search [,replace])
```

功能：STR 为 CHAR、CLOB 和 TEXT 类型，search 和 replace 为字符串类型。在字符串 STR 中找到字符串 search，替换成 replace。若 replace 为空，则在 char 中删除所有 search。

例

```
SELECT  NAME,REPLACE(NAME, '地址', '地点') FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 8.2.4 所示。

表 8.2.4

NAME	REPLACE(NAME, '地址', '地点')
发货地址	发货地点
送货地址	送货地点
家庭地址	家庭地点
公司地址	公司地点

27. 函数 REPLICATE(char,times)

```
语法：REPLICATE(char,times)
```

功能：把字符串 char 自己复制 times 份。

例

```
SELECT  REPLICATE('aaa',3);
```

查询结果为：aaaaaaaaa

28. 函数 REVERSE

```
语法：REVERSE(char)
```

功能：将输入字符串的字符顺序反转后返回。

例：

```
SELECT REVERSE('abcd');
```

查询结果：dcba

29. 函数 RIGHT / RIGHTSTR

```
语法：RIGHT(char,n)  / RIGHTSTR(char,n)
```

功能：返回字符串最右边 n 个字符组成的字符串。

例

```
SELECT  NAME, RIGHT (NAME,2) FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 8.2.5 所示。

表 8.2.5

NAME	RIGHT (NAME,2)
发货地址	地址
送货地址	地址

家庭地址	地址
公司地址	地址

```
SELECT RIGHTSTR('computer',3);
```

查询结果为：ter

30. 函数 RPAD

```
语法：RPAD(char1,length[,char2])
```

功能:返回值为字符串 char1 右边增加 char2,总长度达到 length 的字符串,length 为正整数。如果未指定 char2，缺省值为空格。如果 length 的长度比 char1 大，则返回 char1+char2 的前(length-length(char1))个字符,总长度为 length。如果 length 比 char1 小，则返回 char1 的前 lengh 个字符。长度以字节作为计算单位，一个汉字作为二个字节计算。

注：若 length 为小于或等于零的整数，则返回 null。

例

```
SELECT RPAD('FUXIN',11, 'BigBig') "RPAD example";
```

查询结果为：FUXINBigBig

```
SELECT RPAD('计算机',8, '我们的');
```

查询结果为：计算机我

31. 函数 RTRIM

```
语法：RTRIM(char1[,set])
```

功能：删除字符串 char1 右边起出现的 set 中的任何字符，当遇到不在 set 中的第一个字符时结果被返回。set 缺省为空格。

例

```
SELECT RTRIM('TURNERyXxxxyyyxy', 'xy') "RTRIM e.g.";
```

查询结果为：TURNERyX

```
SELECT RTRIM('我们的计算机','我计算机');
```

查询结果为：我们的

32. 函数 SOUNDEX

```
语法：SOUNDEX(char)
```

功能：返回一个表示英文字符串发音的字符串，由四个字符构成，第一个为英文字符，后三个为数字。NULL返回NULL，当INI参数COMPATIBLE_MODE=0或2时，将忽略原字符串中所有非英文字符，若原字符串为空串或者不存在英文字符则返回NULL；当COMPATIBLE_MODE=3时，遇到非英文字符则不再处理后续字符，若原字符串为空串或者不存在英文字符则返回"0000"。

例

```
SELECT SOUNDEX('Hello');
```

查询结果为：H400

33. 函数 SPACE

```
语法：SPACE(n)
```

功能：返回一个包含 n 个空格的字符串。

例

```
SELECT SPACE(5);
```

查询结果为：□□□□□

```
SELECT CONCAT(CONCAT('Hello',SPACE(3)), 'world');
```

查询结果为：Hello□□□world

说明：□表示空格字符

34．函数 STRPOSDEC

```
语法：STRPOSDEC(char)
```

功能：把字符串 char 中最后一个字符的值减一。

例

```
SELECT STRPOSDEC('hello');
```

查询结果为：helln

35．函数 STRPOSDEC

```
语法：STRPOSDEC(char,pos)
```

功能：把字符串 char 中指定位置 pos 上的字符的值减一。

例

```
SELECT STRPOSDEC('hello',3);
```

查询结果为：heklo

36．函数 STRPOSINC

```
语法：STRPOSINC(char)
```

功能：把字符串 char 中最后一个字符的值加一。

例

```
SELECT STRPOSINC ('hello');
```

查询结果为：hellp

37．函数 STRPOSINC

```
语法：STRPOSINC (char,pos)
```

功能：把字符串 char 中指定位置 pos 上的字符的值加一。

例

```
SELECT STRPOSINC ('hello',3);
```

查询结果为：hemlo

38．函数 STUFF

```
语法：STUFF(char1,begin,n,char2)
```

功能：删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 的 begin 所指位置。

例

```
SELECT STUFF('ABCDEFG',1,3, 'OOO');
```

查询结果为：OOODEFG

39．函数 SUBSTR/SUBSTRING

```
语法：SUBSTR(char[,m[,n]]) / SUBSTRING(char[ from m [ for n ]])
```

功能：返回 char 中从字符位置 m 开始的 n 个字符。若 m 为 0，则把 m 就当作 1 对待。若 m 为正数，则返回的字符串是从左边到右边计算的；反之，返回的字符是从 char 的结尾向左边进行计算的。如果没有给出 n，则返回 char 中从字符位置 m 开始的后续子串。如果 n 小于 0，则返回 NULL。如果 m 和 n 都没有给出，返回 char。函数以字符作为计算单位，一个西文字符和一个汉字都作为一个字符计算。

例

```
SELECT NAME,SUBSTRING(NAME FROM 3 FOR 2) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 8.2.6 所示。

表 8.2.6

NAME	SUBSTRING(NAME FROM 3 FOR 2)
红楼梦	梦
水浒传	传
老人与海	与海
射雕英雄传(全四册)	英雄
鲁迅文集(小说、散文、杂文)全两册	文集
长征	
数据结构(C 语言版)(附光盘)	结构
工作中无小事	中无
突破英文基础词汇	英文
噼里啪啦丛书(全 7 册)	啪啦

```
SELECT SUBSTR('我们的计算机',3,4) "Subs";
```

查询结果为：的计算机

40. 函数 SUBSTRB

```
语法：SUBSTRB(string,m[,n])
```

功能：返回 char 中从第 m 字节位置开始的 n 个字节长度的字符串。若 m 为 0，则 m 就当作 1 对待。若 m 为正数，则返回的字符串是从左边到右边计算的；若 m 为负数，返回的字符是从 char 的结尾向左边进行计算的。若 m 大于字符串的长度，则返回空串。如果没有 n，则缺省的长度为整个字符串的长度。如果 n 小于 1，则返回 NULL。

这里假设字符串 string 的长度为 len，如果 n 的值很大，超过 len - m，则返回的子串的长度为 len - m。

如果开始位置 m 不是一个正常的字符的开始位置，那么返回的结果是 k 个空格（k 的值等于下一个有效字符的开始位置和 m 的差），空格后面是有效字符；如果字符串的 m+n-1 的位置不是一个有效的字符，那么就以空格填充。也就是不截断字符。

例

```
SELECT SUBSTRB('达梦数据库有限公司',4,15);
```

查询结果为：□数据库有限公司

说明：□表示空格字符，下同。

字符串前面是一个空格，这是因为字符串'达梦数据库有限公司'的第 4 个字节不是一个完整的字符的开始，因此用空格代替。

```
SELECT SUBSTRB('我们的计算机',3,4) "Subs", LENGTHB( SUBSTRB('我们的计算机',3,4));
```

查询结果为：们的 4

```
SELECT SUBSTRB('ABCDEFG',3,3) "Subs";
```

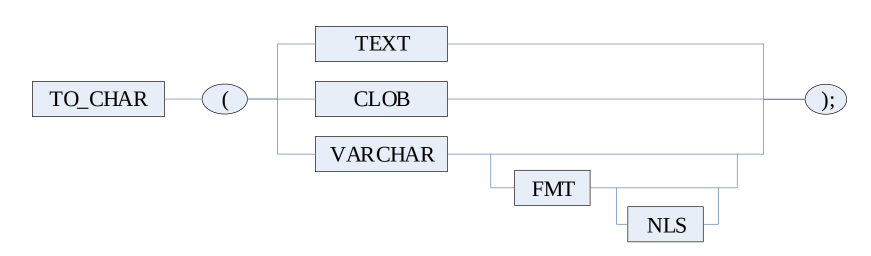
查询结果为：CDE

注意：函数 SUBSTRB 字节作为计算单位，一个字符在不同的编码方式下的字节长度是不同的。

41. 函数 TO_CHAR

```
语法： TO_CHAR(character)
```

图例



功能：将 VARCHAR、CLOB、TEXT 类型的数据转化为 VARCHAR 类型输出。VARCHAR 类型的长度不能超过 32767 个字符，CLOB、TEXT 类型的长度不能超过 32766 个字符。

当参数类型为 VARCHAR 时，还可指定 FMT 与 NLS，FMT 和 NLS 的具体意义和限制可参见 8.1 节的 TO_CHAR 函数介绍。

例

```
SELECT TO_CHAR('0110');
```

查询结果为：0110

```
CREATE TABLE T2(C1 VARCHAR(4000));
INSERT INTO T2 VALUES('达梦数据库有限公司成立于 2000 年，为国有控股的基础软件企业，专业从事数据库管理系统研发、销售和服务。其前身是华中科技大学数据库与多媒体研究所，是国内最早从事数据库管理系统研发的科研机构。达梦数据库为中国数据库标准委员会组长单位，得到了国家各级政府的强力支持。');
SELECT TO_CHAR(C1) FROM T2;
```

查询结果为：达梦数据库有限公司成立于 2000 年，为国有控股的基础软件企业，专业从事数据库管理系统研发、销售和服务。其前身是华中科技大学数据库与多媒体研究所，是国内最早从事数据库管理系统研发的科研机构。达梦数据库为中国数据库标准委员会组长单位，得到了国家各级政府的强力支持。

```
SELECT TO_CHAR('123','99,99','NLS_ISO_CURRENCY=CHINA');
```

查询结果为：1,23

42. 函数 TRANSLATE

```
语法：TRANSLATE(char,from,to)
```

功能：TRANSLATE 是一个字符替换函数。char、from和to分别代表一字符串。对于char字符串，首先，查找char中是否含有from字符串，如果找到，则将其含有的from与to中的字符一一匹配，并用to中相应的字符替换，直至from中的字符全部替换完毕。to中的不足或多余的字符，均视为空值。

例

```
SELECT TRANSLATE('我们的计算机', '我们的', '大世界');
```

查询结果为：大世界计算机 ('我'将被'大'替代,'们'将被'世'替代,'的'将被'

界'替代)

```
SELECT TRANSLATE('我们的计算机', '我们的', '世界');
```

查询结果为：世界计算机　　　('我'将被'世'替代,'们'将被'界'替代,'的'对应的是空值，将被移走)

```
SELECT TRANSLATE('我们的计算机', '我们的', '大大世界');
```

查询结果为：大大世计算机　　　('我'将被'大'替代,'们'将被'大'替代, '的'将被'世'替代, '界'对应的是空值，将被忽略)

例

```
SELECT NAME,TRANSLATE (NAME,'发货','送货') FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 8.2.7 所示。

表 8.2.7

NAME	TRANSLATE (NAME,'发货','送货')
发货地址	送货地址
送货地址	送货地址
家庭地址	家庭地址
公司地址	公司地址

43. 函数 TRIM

```
语法：TRIM([<LEADING|TRAILING|BOTH> [char1] FROM ] char2)
```

功能：TRIM 从 char2 的首端(LEADING)或末端(TRAILING)或两端(BOTH)删除 char1 字符，如果任何一个变量是 NULL，则返回 NULL。默认的修剪方向为 BOTH，默认的修剪字符为空格。

例

```
SELECT  NAME,TRIM(TRAILING '址' FROM NAME) FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 9.2.8 所示。

表 9.2.8

NAME	TRIM(TRAILING '址' FROM NAME)
发货地址	发货地
送货地址	送货地
家庭地址	家庭地
公司地址	公司地

```
SELECT  TRIM( '   Hello World   ');
```

查询结果为：Hello World

```
SELECT  TRIM(LEADING FROM '   Hello World   ');
```

查询结果为：Hello World□□□

说明：□表示空格字符，下同。

```
SELECT  TRIM(TRAILING FROM '   Hello World   ');
```

查询结果为：□□□Hello World

```
SELECT  TRIM(BOTH FROM '   Hello World   ');
```

查询结果为：Hello World

44. 函数 UCASE

```
语法：UCASE(char)
```

功能：返回字符串中，所有字母改为大写，不是字母的字符不受影响。

例

```
SELECT  UCASE('hello world');
```

查询结果为：HELLO WORLD

45. 函数 UPPER

```
语法：UPPER(char)
```

功能：返回字符串中，所有字母改为大写，不是字母的字符不受影响。等价于 UCASE(char)。

46. 函数 REGEXP

REGEXP 函数是根据符合 POSIX 标准的正则表达式进行字符串匹配操作的系统函数，是字符串处理函数的一种扩展。达梦支持的匹配标准如下：

表 8.2.9 符合 POSIX 标准的正则表达式

语法	说明	示例
.	匹配任何除换行符之外的单个字符	d.m 匹配"dameng"
*	匹配前面的字符 0 次或多次	a*b 匹配"bat"中的"b"和"about"中的"ab"。
+	匹配前面的字符一次或多次	ac+ 匹配包含字母"a"和至少一个字母"c"的单词，如"race"和"ace"。
^	匹配行首	^car 仅当单词"car"显示为行中的第一组字符时匹配该单词
$	匹配行尾	end$ 仅当单词"end"显示为可能位于行尾的最后一组字符时匹配该单词
[]	字符集，匹配任何括号间的字符	be[n-t] 匹配"between"中的"bet"、"beneath"中的"ben"和"beside"中的"bes"，但不匹配"below"中的"bel"。
[^]	排除字符集。匹配任何不在括号间的字符	be[^n-t] 匹配"before"中的"bef"、"behind"中的"beh"和"below"中的"bel"，但是不匹配"beneath"中的"ben"。
(表达式)	在表达式加上括号或标签在替换命令中使用	(abc)+匹配"abcabcabc"
|	匹配 OR 符号（|）之前或之后的表达式。最常用在分组中。	(sponge|mud) bath 匹配"sponge bath"和"mud bath"。
\	按原义匹配反斜杠（\）之后的字符。这使您可以查找正则表达式表示法中使用的字符，如 { 和 ^。	\^ 搜索 ^ 字符
{n[,m]}	区间表达式，匹配在它之前的单个字符重现的次数区间。{n}指重复 n 次；{n,}为至少出现 n 次重复；{n,m}为重现 n 至 m 次	zo{2} 匹配"zoone"中的"zoo"，但不匹配"zozo"。
[[:alpha:]]	表示任意字母([a-z]+) | ([A-Z]+)	
[[:digit:]]	表示任意数字\d ([0-9]+)	
[[:lower:]]	表示任意小写字母 ([a-z]+)	
[[:alnum:]]	表示任意字母和数字([a-z0-9]+)	
[[:space:]]	表示任意空格\s	
[[:upper:]]	表示任意大写字母([A-Z]+)	
[[:punct:]]	表示任意标点符号	
[[:xdigit:]]	表示任意 16 进制数([0-9a-fA-F]+)	
\w	表示一个数字或字母字符	

\W	表示一个非数字或字母字符	
\s	表示一个空格字符	
\S	表示一个非空格字符	
\d	表示一个数字字符	
\D	表示一个非数字字符	

值得注意的是，对于 Perl 规则的正则表达式达梦暂不支持：[==],{n}?, \A, \Z, *?, +?, ??, {n}?, {n,}?, {n,m}?。

DM8 支持的 REGEXP 函数如下表：

表 8.2.10 REGEXP 函数

序号	函数名	功能简要说明
1	REGEXP_COUNT(str, pattern[, position [, match_param]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串的个数，并符合匹配参数 match_param
2	REGEXP_LIKE(str, pattern [, match_param])	根据 pattern 正则表达式，查找 str 字符串是否存在符合正则表达式的子串，并符合匹配参数 match_param
3	REGEXP_INSTR(str, pattern[, position[, occurrence [, return_opt [, match_param [, subexpr]]]]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，如果 return_opt 为 0，返回第 occurrence 次出现的位置，如果 return_opt 为大于 0，则返回该出现位置的下一个字符位置，并符合匹配参数。Subexpr 为匹配的子 pattern。
4	REGEXP_SUBSTR(str, pattern [,position [, occurrence [,match_param[, subexpr]]]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，返回第 occurrence 次出现的子串，并符合匹配参数 match_param。
5	REGEXP_REPLACE(str, pattern [, replace_str [, position [, occurrence [,match_param]]]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串，并用 replace_str 进行替换第 occurrence 次出现的子串，并符合匹配参数 match_param。

参数说明：

str：待匹配的字符串，最大长度为 8188 字节；

pattern：符合 POSIX 标准的正则表达式，最大长度为 512 字节；

position：匹配的源字符串的开始位置，正整数，默认为 1；

occurrence：匹配次数，正整数，默认为 1；

match_param：正则表达式的匹配参数，默认大小写敏感，如下表所示：

表 8.2.11 匹配参数

值	说明
c	表示大小写敏感。例如：REGEXP_COUNT('AbCd', 'abcd', 1, 'c')，结果为 0。
i	表示大小写不敏感。例如：REGEXP_COUNT('AbCd', 'abcd', 1, 'i')，结果为 1。
m	将源字符串当成多行处理，默认当成一行。 例如：REGEXP_COUNT('ab'||CHR(10)||'ac', '^a.', 1, 'm')，结果为 2。
n	通配符（.）匹配换行符，默认不匹配。 例如：REGEXP_COUNT('a'||CHR(10)||'d', 'a.d', 1, 'n')，结果为 1。
x	忽略空格字符。例如：REGEXP_COUNT('abcd', 'a b c d', 1, 'x')，结果为 1。

return_opt：正整数，返回匹配子串的位置。值为 0：表示返回子串的开始位置；值大于 0：表示返回子串结束位置的下一个字符位置；

subexpr：正整数，取值范围为：0~9，表示匹配第 subexpr 个子 pattern 正则表

达式，子 pattern 必须是括号的一部分。如果 subexpr=0，则表示匹配整个正则表达式；如果 subexpr > 0，则匹配对应的第 subexpr 个子 pattern；如果 subexpr 大于子 pattern 个数，则返回 0；如果 subexpr 为 NULL，则返回 NULL。

replace_str：用于替换的字符串，最大长度为 512 字节。

DM 的 REGEXP 函数支持正则表达式的反向引用，通过"\数字"的方式进行引用，如\1 表示第一个匹配的子表达式。

如下详细介绍各函数：

1． 函数 REGEXP_COUNT

```
语法：REGEXP_COUNT(str, pattern[, position [, match_param]])
```

功能：根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串的个数，并符合匹配参数 match_param。position 默认值为 1，position 为正整数，小于 0 则报错；如果 position 为空，则返回 NULL。pattern 必须符合正则表达式的规则，否则报错。match_param 不合法，则报错。

返回值：如果 str、pattern 和 match_param 其中有一个为空串或 NULL，则返回 NULL。如果不匹配，返回 0；如果匹配，返回匹配的个数。

例

```
SELECT REGEXP_COUNT('AbCd', 'abcd', 1, 'i') FROM DUAL;
```

查询结果：1

```
SELECT REGEXP_COUNT('AbCd', 'abcd', 1, 'c') FROM DUAL;
```

查询结果：0

2． 函数 REGEXP_LIKE

```
语法：REGEXP_LIKE(str, pattern [, match_param])
```

功能：根据 pattern 正则表达式，查找 str 字符串是否存在符合正则表达式的子串，并符合匹配参数 match_param。

返回值：如果匹配，则返回 TRUE；否则返回 FALSE。如果 str、pattern 和 match_param 中任一个为空串或 NULL，则返回 NULL；

例

```
SELECT 1 FROM DUAL WHERE REGEXP_LIKE('DM database V7', 'dm', 'c');
```

查询结果：无返回行

```
SELECT 1 FROM DUAL WHERE REGEXP_LIKE('DM database V7', 'dm', 'i');
```

查询结果：1

3． 函数 REGEXP_INSTR

```
语法：REGEXP_INSTR(str, pattern[, position[,  occurrence [, return_opt [, match_param [, subexpr]]]]])
```

功能：根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，如果 return_opt 为 0，返回第 occurrence 次出现的位置，如果 return_opt 为大于 0，则返回该出现位置的下一个字符位置，并符合匹配参数。Subexpr 为匹配的子 pattern。

返回值：如果 str、pattern、position、occurrence、return_opt、match_param 和 subexpr 中任一个为 NULL，则返回 NULL。否则返回符合条件的子串位置，如果没有找到，则返回 0。

例

```
SELECT REGEXP_INSTR('a为了aaac','aa') FROM DUAL;
```

查询结果：4

```
SELECT REGEXP_INSTR('a为了aaac','aa',5) FROM DUAL;
```

查询结果：5

```
SELECT REGEXP_INSTR('1234567890', '(123)(4(56)(78))', 1, 1, 0, 'i', 2)
"REGEXP_INSTR" FROM DUAL;
```

查询结果：4

4. REGEXP_SUBSTR

```
语法：REGEXP_SUBSTR(str, pattern [,position [, occurrence [,match_param[,
subexpr]]]])
```

功能：根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，返回第 occurrence 次出现的子串，并符合匹配参数 match_param。occurrence 默认为 1。如果 position 或 occurrence 的输入值不为正数，则报错。

返回值：如果 str、pattern、position、occurrence、match_param 和 subexpr 中任一个为 NULL，则返回 NULL。如果找到符合正则表达式的子串，则返回匹配的子串；如果没有找到，则返回 NULL。

例

```
SELECT REGEXP_SUBSTR('a为aa了aac','(a*)',2) FROM DUAL;
```

查询结果：空

```
SELECT REGEXP_SUBSTR('a为aa了aac','(a+)',2) FROM DUAL;
```

查询结果：aa

```
SELECT REGEXP_SUBSTR('500 DM8 DATABASE, SHANG HAI, CN', ',[^,]+,', 5, 1, 'i',
0) "REGEXPR_SUBSTR"  FROM DUAL;
```

查询结果：, SHANG HAI,

5. REGEXP_REPLACE

```
语法：REGEXP_REPLACE(str, pattern [, replace_str [, position [, occurrence
[,match_param]]]])
```

功能：根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串，并用 replace_str 进行替换第 occurrence 次出现的子串，并符合匹配参数 match_param。occurrence 默认为 0，替换所有出现的子串。replace_str 默认为空串，在替换过程中，则相当于删除查找到的子串； position 默认值为 1，如果 position 的值不为正整数，则报错；

返回值：返回替换后的 str。如果 str、pattern、position、occurrence 和 match_param 中任一个为 NULL，则返回 NULL；如果 str 中所有的字符都被空串替换，则返回 NULL，相当于删除所有的字符。

例

```
SELECT REGEXP_REPLACE('a为了aaac','aa','bb') FROM DUAL;
```

查询结果：a为了bbac

```
SELECT REGEXP_REPLACE('a为了ac','aa','bb') FROM DUAL;
```

查询结果：a为了ac

```
SELECT REGEXP_REPLACE('a为aa了aac','aa','bb') FROM DUAL;
```

查询结果：a为bb了bbc

```
SELECT REGEXP_REPLACE('500 DM8 DATABASE, SHANG HAI, CN', ',[^,]+,', ', WU
HAN,', 5, 1,'i') "REGEXPR_REPLACE"  FROM DUAL;
```

查询结果：500 DM8 DATABASE, WU HAN, CN

```
SELECT REGEXP_REPLACE('www1234xxxx3q', '([[:alpha:]]+)', 'AAA\1') FROM
```

```
DUAL;
```

此处使用了正则表达式的反向引用功能，查询结果为：AAAwww1234AAAxxxx3AAAq

47. 函数 OVERLAY

```
语法：OVERLAY(char1 PLACING char2 FROM m [ FOR n ])
```

功能：用串 char2（称为"替换字符串"）覆盖源串 char1 的指定子串，该子串是通过在源串中的给定起始位置的数值（m）和长度的数值（n）而指明，来修改一个串自变量。当子串长度为 0 时，不会从源串中移去任何串；当不指定 n 时，默认 n 为 char2 的长度。函数的返回串是在源串的给定起始位置插入替换字符串所得的结果。

例

```
SELECT NAME,OVERLAY(NAME PLACING '口' FROM 3 FOR 2) FROM PRODUCTION.PRODUCT;
```

查询结果如下表所示。

表 8.2.12

NAME	"OVERLAY"(NAME,'口',3,2)
红楼梦	红楼口
水浒传	水浒口
老人与海	老人口
射雕英雄传(全四册)	射雕口传(全四册)
鲁迅文集(小说、散文、杂文)全两册	鲁迅口(小说、散文、杂文)全两册
长征	长征口
数据结构(C 语言版)(附光盘)	数据口(C 语言版)(附光盘)
工作中无小事	工作口小事
突破英文基础词汇	突破口基础词汇
噼里啪啦丛书(全 7 册)	噼里口丛书(全 7 册)

```
SELECT OVERLAY('txxxxas' PLACING 'hom' FROM 2 FOR 4);
```

查询结果为：thomas

48. 函数 TEXT_EQUAL

```
语法：TEXT_EQUAL(n1,n2)
```

功能：返回 n1，n2 的比较结果，完全相等，返回 1；否则返回 0。n1，n2 的类型为 CLOB、TEXT 或 LONGVARCHAR。如果 n1 或 n2 均为空串或 NULL，结果返回为 1；否则只有一个为空串或为 NULL，结果返回 0。不忽略结果空格和英文字母大小写。

例

```
SELECT TEXT_EQUAL('a', 'b');
```

查询结果为：0

```
SELECT TEXT_EQUAL('a','a');
```

查询结果为：1

49. 函数 BLOB_EQUAL

```
语法：BLOB_EQUAL(n1,n2)
```

功能：返回 n1，n2 两个数的比较结果，完全相等，返回 1；否则返回 0。n1，n2 的类型为 BLOB、IMAGE 或 LONGVARBINARY。如果 n1 或 n2 均为空串或 NULL，结果返回为 1；否则只有一个为空串或为 NULL，结果返回 0。

例

```
SELECT BLOB_EQUAL(0xFFFEE, 0xEEEFF);
```

查询结果为：0

```
SELECT BLOB_EQUAL(0xFFFEE, 0xFFFEE);
```

查询结果为：1

50．函数 NLSSORT

```
语法：NLSSORT(str1 [,nls_sort=str2])
```

功能：返回对汉字排序的编码。当只有 str1 一个参数时，与 RAWTOHEX 类似，返回 16 进制字符串。str2 决定按哪种方式排序，可以选择 schinese_pinyin_m、schinese_stroke_m、schinese_radical_m。分别表示支持对汉字按拼音、笔画、部首排序。

例

```
SELECT NLSSORT('abc') FROM DUAL;
```

查询结果为：61626300

```
CREATE TABLE TEST(C1 VARCHAR2(200));
INSERT INTO TEST VALUES('啊');
INSERT INTO TEST VALUES('不');
INSERT INTO TEST VALUES('才');
INSERT INTO TEST VALUES('的');
INSERT INTO TEST VALUES('一');
INSERT INTO TEST VALUES('二');
INSERT INTO TEST VALUES('三');
INSERT INTO TEST VALUES('四');
INSERT INTO TEST VALUES('品');
INSERT INTO TEST VALUES('磊');
SELECT * FROM TEST ORDER BY NLSSORT(C1, 'NLS_SORT=SCHINESE_PINYIN_M');
--拼音
```

查询结果为：

```
行号        C1
---------- --
1          啊
2          不
3          才
4          的
5          二
6          磊
7          品
8          三
9          四
10         一
```

```
SELECT * FROM TEST ORDER BY NLSSORT(C1, 'NLS_SORT=SCHINESE_STROKE_M');
--笔画
```

查询结果为：

```
行号        C1
---------- --
1           一
2           二
3           三
4           才
5           不
6           四
7           的
8           品
9           啊
10          磊
```

```
SELECT * FROM TEST ORDER BY NLSSORT(C1, 'NLS_SORT=SCHINESE_RADICAL_M');
--部首
```

查询结果为：

```
行号        C1
---------- --
1           一
2           二
3           三
4           不
5           品
6           啊
7           四
8           才
9           的
10          磊
```

```
SELECT C1,NLSSORT(C1),NLSSORT(C1, 'NLS_SORT=SCHINESE_PINYIN_M') FROM TEST
ORDER BY NLSSORT(C1, 'NLS_SORT=SCHINESE_PINYIN_M');
```

分别返回 c1，返回将 c1 转化后的 16 进制字符串，返回用来为汉字排序的编码。

查询结果为：

```
C1    NLSSORT ( C1 )    NLSSORT ( C1, 'nls_sort=schinese_pinyin_m')
 啊    B0A100   3B2C
 不    B2BB00   4248
 才    B2C500   4291
 的    B5C400   4D8D
 二    B6FE00   531D
 磊    C0DA00   743E
 品    C6B700   8898
 三    C8FD00   932C
 四    CBC400   996A
 一    D2BB00   B310
```

51. 函数 GREATEST

```
语法：GREATEST(char 1, char 2, char 3)
```

功能：求 char 1、char 2 和 char 3 中最大的字符串。

例

```
SELECT GREATEST('abb','abd', 'abc');
```

查询结果：abd

52. 函数 GREAT

```
语法：GREAT (char 1, char 2)
```

功能：求 char 1、char 2 中最大的字符串。

例

```
SELECT GREAT ('abb','abd');
```

查询结果：abd

53. 函数 TO_SINGLE_BYTE

```
语法： TO_SINGLE_BYTE(
        STR IN VARCHAR
     ) return RVAL VARCHAR;
```

功能：将多字节形式的字符（串）转换为对应的单字节形式
参数： STR：待转换的字符，长度为 4096
返回值： RVAL：函数返回值，长度为 8187
例：

```
SELECT LENGTHB(TO_SINGLE_BYTE('aa'));
```

返回结果：2

54. 函数 TO_MULTI_BYTE

```
语法： to_multi_byte (
        STR IN VARCHAR
     ) return RVAL VARCHAR;
```

功能：将单字节形式的字符（串）转换为对应的多字节形式 (不同的字符集转换结果不同)
参数：STR ：待转换的字符，长度为 4096
返回值：RVAL：函数返回值，长度为 8187
例

```
SELECT LENGTHB(TO_MULTI_BYTE('aa'));
```

返回结果：4

55. 函数 EMPTY_BLOB

```
语法：EMPTY_BLOB return RVAL blob;
```

功能：初始化blob字段

返回值：RVAL：长度为0的blob数据

例

```
DROP TABLE TT;
CREATE TABLE TT(C1 BLOB, C2 INT);
INSERT INTO TT VALUES(EMPTY_BLOB(),1);
```

```
INSERT INTO TT VALUES(NULL,2);
INSERT INTO TT VALUES(0X123,3);
SELECT LENGTHB(C1) FROM TT;
```

返回值：

```
 LENGTHB(C1)
-----------
0
NULL
2
```

56. 函数 EMPTY_CLOB

```
语法：EMPTY_CLOB return RVAL clob;
```

功能：初始化clob字段

返回值： RVAL：长度为0的clob数据

例

```
DROP TABLE TT;
CREATE TABLE TT(C1 CLOB, C2 INT);
INSERT INTO TT VALUES(EMPTY_CLOB(),1);
INSERT INTO TT VALUES(NULL,2);
INSERT INTO TT VALUES('0X123',3);
SELECT LENGTHB(C1) FROM TT;
```

返回值：

```
LENGTHB(C1)
-----------
0
NULL
5
```

57. 函数 UNISTR

```
语法：UNISTR (char)
```

功能：将字符串char中，ascii码（'\XXXX'4个16进制字符格式）转成本地字符。对于其他字符保持不变。

例 在 GB18030 库下，执行如下操作：

```
SELECT UNISTR('\803F\55B5\55B5kind 又\006e\0069\0063\0065') FROM DUAL;
```

查询结果为： 耿喵喵kind又nice

58. 函数 ISNULL

```
语法：ISNULL(char)
```

功能：判断表达式是否为NULL，为NULL返回1，否则返回0。

例 查询总经理的 MANAGERID 是否为空：

```
SELECT ISNULL(MANAGERID) FROM RESOURCES.EMPLOYEE WHERE TITLE='总经理';
```

8.3 日期时间函数

日期时间函数的参数至少有一个是日期时间类型(TIME，DATE，TIMESTAMP)，返回值一般为日期时间类型和数值类型。

由于 DM 支持儒略历，并考虑了历史上从儒略历转换至格里高利日期时的异常，不计算'1582-10-05'到'1582-10-14'之间的 10 天，因此日期时间函数也不计算这 10 天。

1. 函数 ADD_DAYS

```
语法：ADD_DAYS( date, n)
```

功能：返回日期 date 加上相应天数 n 后的日期值。n 可以是任意整数，date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回值为日期类型(DATE)。

例

```
SELECT ADD_DAYS( DATE '2000-01-12',1);
```

查询结果为：2000-01-13

2. 函数 ADD_MONTHS

```
语法：ADD_MONTHS(date,n)
```

功能：返回日期 date 加上 n 个月的日期时间值。n 可以是任意整数，date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回类型固定为日期类型(DATE)。如果相加之后的结果日期中月份所包含的天数比 date 日期中的日分量要少，那么结果日期的该月最后一天被返回。

例

```
SELECT ADD_MONTHS(DATE '2000-01-31',1);
```

查询结果为：2000-02-29

```
SELECT ADD_MONTHS(TIMESTAMP '2000-01-31 20:00:00',1);
```

查询结果为：2000-02-29

3. 函数 ADD_WEEKS

```
语法：ADD_WEEKS( date, n)
```

功能：返回日期 date 加上相应星期数 n 后的日期值。n 可以是任意整数，date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回类型固定为日期类型(DATE)。

例

```
SELECT ADD_WEEKS( DATE '2000-01-12',1);
```

查询结果为： 2000-01-19

4. 函数 CURDATE

```
语法：CURDATE()
```

功能：返回当前日期值，结果类型为 DATE。

例

```
SELECT   CURDATE();
```

查询结果为：执行此查询当天日期，如 2003-02-27

5. 函数 CURTIME

```
语法：CURTIME(n)
```

功能：返回当前时间值，结果类型为 TIME WITH TIME ZONE。
参数：n：指定毫秒的精度。取值范围 0-6，默认为 6。
例

```
SELECT  CURTIME();
```

查询结果为：执行此查询的当前时间，如 14:53:54.859000 +8:00

6. 函数 CURRENT_DATE

```
语法：CURRENT_DATE
```

功能：返回当前日期值，结果类型为 DATE，等价于 CURDATE()。

7. 函数 CURRENT_TIME

```
语法：CURRENT_TIME(n)
```

功能：返回当前时间值，结果类型为 TIME WITH TIME ZONE，等价于 CURTIME()。
参数：n：指定毫秒的精度。取值范围 0-6，默认为 6。

8. 函数 CURRENT_TIMESTAMP

```
语法：CURRENT_TIMESTAMP(n)
```

功能：返回当前带会话时区的时间戳，结果类型为 TIMESTAMP WITH TIME ZONE。
参数：n：指定毫秒的精度。取值范围 0-6，默认为 6。
例

```
SELECT   CURRENT_TIMESTAMP();
```

查询结果为：执行此查询的当前日期时间，如 2011-12-27 13:03:56.000000 +8:00

9. 函数 DATEADD

```
语法：DATEADD(datepart,n,date)
```

功能：向指定的日期 date 加上 n 个 datepart 指定的时间段，返回新的 timestamp 值。datepart 取值见下表。

表 8.3.1 datepart 取值

datepart 取值	datepart 意义
YEAR、YYYY、YY、SQL_TSI_YEAR	年
MONTH、MM、M、SQL_TSI_MONTH	月
DAY、DD、D、SQL_TSI_DAY	日
HOUR、HH、SQL_TSI_HOUR	时
MINUTE、MI、N、SQL_TSI_MINUTE	分
SECOND、S、SQL_TSI_SECOND	秒
MILLISECOND、MS、SQL_TSI_FRAC_SECOND	毫秒
QUARTER、QQ、Q、SQL_TSI_QUARTER	所处的季度
DAYOFYEAR、DY、Y	在年份中所处的天数
WEEK、WK、WW、SQL_TSI_WEEK	在年份中所处的周数
WEEKDAY、DW	在一周中所处的天数

10. 函数 DATEDIFF/BIGDATEDIFF

```
语法：DATEDIFF(datepart,date1,date2)
```

功能：返回跨两个指定日期的日期和时间边界数。datepart 取值见表 8.3.1。

注：当结果超出整数值范围，DATEDIFF 产生错误。对于毫秒 MILLISECOND，最大数是 24 天 20 小时 31 分钟零 23.647 秒。对于秒，最大数是 68 年。若想提高可以表示的范围，可以使用 BIGDATEDIFF，其使用方法与 DATEDIFF 函数一致，只是可以表示更广范围的秒和毫秒。

例

```
SELECT DATEDIFF(QQ, '2003-06-01', DATE '2002-01-01');
```

查询结果为：-5

```
SELECT DATEDIFF(MONTH, '2001-06-01', DATE '2002-01-01');
```

查询结果为：7

```
SELECT DATEDIFF(WK, DATE '2003-02-07',DATE '2003-02-14');
```

查询结果为：1

```
SELECT DATEDIFF(MS,'2003-02-14 12:10:10.000','2003-02-14 12:09:09.300');
```

查询结果为：-60700

11．函数 DATEPART/DATE_PART

```
语法：DATEPART(datepart,date)
```

功能：返回代表日期 date 的指定部分的整数。datepart 取值请参考 DATEDIFF(datepart,date1,date2)的参数。

例

```
SELECT DATEPART(SECOND, DATETIME '2000-02-02 13:33:40.00');
```

查询结果为：40

```
SELECT DATEPART(DY, '2000-02-02');
```

查询结果为：33

```
SELECT DATEPART(WEEKDAY, '2002-02-02');
```

查询结果为：7

说明：日期函数：date_part，其功能与 datepart 完全一样。但是写法有点不同：select datepart(year,'2008-10-10');如果用 date_part，则要写成：select date_part('2008-10-10','year')，即：参数顺序颠倒，同时指定要获取的日期部分的参数要带引号。

12．函数 DAY

```
语法：DAY(date)
```

功能：返回指定日期在月份中的天数

例

```
SELECT DAY('2016-06-07');
```

查询结果为：7

13．函数 DAYNAME

```
语法：DAYNAME(date)
```

功能：返回日期的星期名称。

例

```
SELECT DAYNAME(DATE '2012-01-01');
```

查询结果为：Sunday

14．函数 DAYOFMONTH

```
语法：DAYOFMONTH(date)
```

功能：返回日期为所处月份中的第几天。

例

```
SELECT DAYOFMONTH('2003-01-03');
```

查询结果为：3

15．函数 DAYOFWEEK

```
语法：DAYOFWEEK(date)
```

功能：返回日期为所处星期中的第几天。

例

```
SELECT DAYOFWEEK('2003-01-01');
```

查询结果为：4

16．函数 DAYOFYEAR

```
语法：DAYOFYEAR(date)
```

功能：返回日期为所处年中的第几天。

例

```
SELECT DAYOFYEAR('2003-03-03');
```

查询结果为：62

17．函数 DAYS_BETWEEN

```
语法： DAYS_BETWEEN(dt1,dt2)
```

功能：返回两个日期之间相差的天数。

18．函数 EXTRACT

```
语法：EXTRACT(dtfield FROM date)
```

功能:EXTRACT 从日期时间类型或时间间隔类型的参数 date 中抽取 dtfield 对应的数值，并返回一个数字值。如果 date 是 NULL，则返回 NULL。Dtfiled 可以是 YEAR、MONTH、DAY、HOUR、MINUTE、SECOND。对于 SECOND 之外的任何域，函数返回整数，对于 SECOND 返回小数。

例：

```
SELECT EXTRACT(YEAR FROM DATE '2000-01-01');
```

查询结果为：2000

```
SELECT EXTRACT(DAY FROM DATE '2000-01-01');
```

查询结果为：1

```
SELECT EXTRACT(MINUTE FROM TIME '12:00:01.35');
```

查询结果为：0

```
SELECT EXTRACT(TIMEZONE_HOUR FROM TIME '12:00:01.35 +9:30');
```

查询结果为：9

```
SELECT EXTRACT(TIMEZONE_MINUTE FROM TIME '12:00:01.35 +9:30');
```

查询结果为：30

```
SELECT EXTRACT(SECOND FROM TIMESTAMP '2000-01-01 12:00:01.35');
```

查询结果为：1.3500000000E+000

```
SELECT EXTRACT(SECOND FROM INTERVAL '-05:01:22.01' HOUR TO SECOND);
```

查询结果为：-2.2010000000E+001

19．函数 GETDATE

```
语法：GETDATE(n)
```

功能：返回系统的当前时间戳。

参数：n：指定毫秒的精度。取值范围 0-6，默认为 6。

例

```
SELECT GETDATE();
```

查询结果为：返回系统的当前日期时间，如 2011-12-05 11:31:10.359000

20．函数 GREATEST

```
语法：GREATEST(date1,date2,date3)
```

功能：求 date1、date2 和 date3 中的最大日期。

例

```
SELECT GREATEST(date'1999-01-01',date'1998-01-01',date'2000-01-01');
```

查询结果：2000-01-01

21．函数 GREAT

```
语法：GREAT (date1,date2)
```

功能：求 date1、date2 中的最大日期。

例

```
SELECT GREAT (date'1999-01-01', date'2000-01-01');
```

查询结果：2000-01-01

22．函数 HOUR

```
语法：HOUR(time)
```

功能：返回时间中的小时分量。

例

```
SELECT HOUR(TIME '20:10:16');
```

查询结果为：20

23．函数 LAST_DAY

```
语法：LAST_DAY(date)
```

功能：返回 date 所在月最后一天的日期,date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回类型与 date 相同。

例

```
SELECT LAST_DAY(SYSDATE) "Days Left";
```

查询结果为：如：当前日期为 2003 年 2 月的某一天，则结果为 2003-02-28

```
SELECT LAST_DAY(TIMESTAMP '2000-01-11 12:00:00');
```

查询结果为：2000-01-31

24．函数 LEAST

```
语法：LEAST(date1,date2,date3)
```

功能：求date1、date2和date3中的最小日期。

例

```
SELECT LEAST(date'1999-01-01',date'1998-01-01',date'2000-01-01');
```

查询结果：1998-01-01

25. 函数 MINUTE

```
语法：MINUTE(time)
```

功能：返回时间中的分钟分量。

例

```
SELECT MINUTE('20:10:16');
```

查询结果为：10

26. 函数 MONTH

```
语法：MONTH(date)
```

功能：返回日期中的月份分量。

例

```
SELECT MONTH('2002-11-12');
```

查询结果为：11

27. 函数 MONTHNAME

```
语法：MONTHNAME(date)
```

功能：返回日期中月份分量的名称。

例

```
SELECT MONTHNAME('2002-11-12');
```

查询结果为：November

28. 函数 MONTHS_BETWEEN

```
语法：MONTHS_BETWEEN(date1,date2)
```

功能：返回 date1 和 date2 之间的月份值。如果 date1 比 date2 晚，返回正值，否则返回负值。如果 date1 和 date2 这两个日期为同一天，或者都是所在月的最后一天，则返回整数，否则返回值带有小数。date1 和 date2 是日期类型(DATE)或时间戳类型(TIMESTAMP)。

例

```
SELECT MONTHS_BETWEEN(DATE '1995-02-28', DATE '1995-01-31') "Months";
```

查询结果为：1.0

```
SELECT MONTHS_BETWEEN(TIMESTAMP '1995-03-28 12:00:00',
TIMESTAMP '1995-01-31 12:00:00') "Months";
```

查询结果为：1.903226

29. 函数 NEXT_DAY

```
语法：NEXT_DAY(date,char)
```

功能：返回在日期 date 之后满足由 char 给出的条件的第一天。char 指定了一周中的某一个天(星期几)，返回值的时间分量与 date 相同，char 是大小写无关的。

Char 取值如表 8.3.8 所示。

表 8.3.2 星期描述说明

输入值	含义
SUN	星期日
SUNDAY	
MON	星期一
MONDAY	
TUES	星期二
TUESDAY	
WED	星期三
WEDNESDAY	
THURS	星期四
THURSDAY	
FRI	星期五
FRIDAY	
SAT	星期六
SATURDAY	

例

```
SELECT NEXT_DAY(DATE '2001-08-02', 'MONDAY');
```

查询结果为：2001-08-06

```
SELECT NEXT_DAY('2001-08-02 12:00:00', 'FRI');
```

查询结果为：2001-08-03

30．函数 NOW

```
语法：NOW(n)
```

功能：返回系统的当前时间戳。等价于 GETDATE()。

参数：n：指定毫秒的精度。取值范围 0-6，默认为 6。

31．函数 QUARTER

```
语法：QUARTER(date)
```

功能：返回日期在所处年中的季度数。

例

```
SELECT QUARTER('2002-08-01');
```

查询结果为：3

32．函数 SECOND

```
语法：SECOND(time)
```

功能：返回时间中的秒分量。

例

```
SELECT SECOND('08:10:25.300');
```

查询结果为：25

33．函数 ROUND

```
语法：ROUND(date[, fmt])
```

功能：将日期时间 date 四舍五入到最接近格式参数 fmt 指定的形式。如果没有指定语法的话，到今天正午 12P.M.为止的时间舍取为今天的日期，之后的时间舍取为第二天

12A.M.。日期时间 12A.M.，为一天的初始时刻。参数 date 的类型可以是 DATE 或 TIMESTAMP，但应与 fmt 相匹配。函数的返回结果的类型与参数 date 相同。fmt 具体如表 8.3.8 所示。

表 8.3.3 日期时间说明

fmt 的格式	**含义**	**date 数据类型**
cc，scc	世纪，从 1950、2050 等年份的一月一号午夜凌晨起的日期，舍取至下个世纪的一月一号	DATE TIMESTAMP
syear，syyy，y，yy，yyy，yyyy，year	年，从七月一号午夜凌晨起的日期，舍取至下个年度的一月一号	DATE TIMESTAMP
Q	季度，从十六号午夜凌晨舍取到季度的第二个月，忽略月中的天数	DATE TIMESTAMP
month,mon，mm，m，rm	月，从十六号午夜凌晨舍取	DATE TIMESTAMP
Ww	舍取为与本年第一天星期数相同的最近的那一天	DATE TIMESTAMP
W	舍取为与本月第一天星期数相同的最近的一天	DATE TIMESTAMP
iw	舍取为最近的周一	DATE TIMESTAMP
ddd，dd，j	从正午起，舍取为下一天，默认值	DATE TIMESTAMP
day，dy，d	星期三正午起，舍取为下个星期天	DATE TIMESTAMP
hh，hh12，hh24	在一个小时的 30 分 30 秒之后的时间舍取为下一小时	TIME TIMESTAMP
Mi	在一个分钟 30 秒之后的时间舍取为下一分	TIME TIMESTAMP

有关 ww 和 w 的计算进一步解释如下(下面的时间仅当 date 参数为时间戳时才有效)：

ww 产生与本年第一天星期数相同的最近的日期。因为每两个星期数相同日期之间相隔六天，这意味着舍取结果在给定日期之后三天以内。例如，如果本年第一天为星期二，若给定日期在星期五午夜 23:59:59 之前(包含星期五 23:59:59)，则舍取为本星期的星期二的日期；否则舍取为下星期的星期二的日期。

w 计算的方式类似，不是产生最近的星期一 00:00:00，而是产生与本月第一天相同的星期数的日期。

例

```
SELECT ROUND(DATE '1992-10-27', 'scc');
```

查询结果为：2001-01-01

```
SELECT ROUND(DATE '1992-10-27', 'YEAR') "FIRST OF THE YEAR";
```

查询结果为：1993-01-01

```
SELECT ROUND(DATE '1992-10-27', 'q');
```

查询结果为：1992-10-01

```
SELECT ROUND(DATE '1992-10-27', 'month');
```

查询结果为：1992-11-01

```
SELECT ROUND(TIMESTAMP '1992-10-27 11:00:00', 'ww');
```

查询结果为：1992-10-28 00:00:00.000000

```
SELECT ROUND(TIMESTAMP '1992-10-27 11:00:00', 'w');
```

查询结果为：1992-10-29 00:00:00.000000

```
SELECT ROUND(TIMESTAMP '1992-10-27 12:00:01', 'ddd');
```

查询结果为：1992-10-28 00:00:00.000000

```
SELECT ROUND(DATE '1992-10-27', 'day');
```

查询结果为：1992-10-25

```
SELECT ROUND(TIMESTAMP '1992-10-27 12:00:31', 'hh');
```

查询结果为：1992-10-27 12:00:00.000000

```
SELECT ROUND(TIMESTAMP '1992-10-27 12:00:31', 'mi');
```

查询结果为：1992-10-27 12:01:00.000000

34. 函数 TIMESTAMPADD

```
语法：TIMESTAMPADD(datepart,n,timestamp)
```

功能：返回时间戳 timestamp 加上 n 个 datepart 指定的时间段的结果，datepart 取值见表 8.3.1。

例

```
SELECT TIMESTAMPADD(SQL_TSI_FRAC_SECOND, 5, '2003-02-10 08:12:20.300' );
```

查询结果为：2003-02-10 08:12:20.305000

```
SELECT TIMESTAMPADD(SQL_TSI_YEAR, 30, DATE '2002-01-01');
```

查询结果为：2032-01-01 00:00:00.000000

```
SELECT TIMESTAMPADD(SQL_TSI_QUARTER, 2, TIMESTAMP '2002-01-01 12:00:00');
```

查询结果为：2002-07-01 12:00:00.000000

```
SELECT TIMESTAMPADD(SQL_TSI_DAY, 40, '2002-12-01 12:00:00');
```

查询结果为：2003-01-10 12:00:00.000000

```
SELECT TIMESTAMPADD(SQL_TSI_WEEK, 1, '2002-01-30');
```

查询结果为：2002-02-06 00:00:00.000000

35. 函数 TIMESTAMPDIFF

```
语法：TIMESTAMPDIFF(datepart,timestamp1,timestamp2)
```

功能：返回一个表明 timestamp2 与 timestamp1 之间的指定 datepart 类型的时间间隔的整数，datepart 取值见表 8.3.1。

注：当结果超出整数值范围，TIMESTAMPDIFF 产生错误。对于秒级 SQL_TSI_SECOND，最大数是 68 年。

例

```
SELECT TIMESTAMPDIFF(SQL_TSI_FRAC_SECOND,
'2003-02-14 12:10:10.000', '2003-02-14 12:09:09.300');
```

查询结果为：-60700

```
SELECT TIMESTAMPDIFF(SQL_TSI_QUARTER, '2003-06-01', DATE '2002-01-01');
```

查询结果为：-5

```
SELECT TIMESTAMPDIFF(SQL_TSI_MONTH, '2001-06-01', DATE '2002-01-01');
```

查询结果为：7

```
SELECT TIMESTAMPDIFF(SQL_TSI_WEEK, DATE '2003-02-07',DATE '2003-02-14');
```

查询结果为：1

36. 函数 SYSDATE

```
语法： SYSDATE()
```

功能： 获取系统当前时间。

例

```
SELECT  SYSDATE();
```

查询结果：当前系统时间

37. 函数 TO_DATE/TO_TIMESTAMP/TO_TIMESTAMP_TZ

```
语法：TO_DATE(char [,fmt[,'nls']])  或 TO_TIMESTAMP(char [,fmt[,'nls']]) 或
TO_TIMESTAMP_TZ(char [,fmt])
```

功能:将 CHAR 或者 VARCHAR 类型的值转换为 DATE/TIMESTAMP 数据类型。TO_DATE 的结果不带毫秒精度，TO_TIMESTAMP 的结果带 6 位毫秒精度。TO_TIMESTAMP_TZ 的结果带上服务器的时区。

NLS：指定日期时间串的语言类型：美语，英语和简体中文。 这个参数的形式是："NLS_DATE_LANGUAGE=语言类型"，可以有这几种取值： AMERICAN、ENGLISH 和 SIMPLIFIED CHINESE。其中，AMERICAN 和 ENGLISH 的效果相同。缺省为 SIMPLIFIED CHINESE。

FMT：指定日期语法格式，需按照特定的格式书写。FMT 内容有三种选择：DATE、DATE+TIME 或 TIME。合法的 DATE 格式为年月日、月日年或日月年，各部分之间可以有分隔符或者没有分隔符；合法的 TIME 格式为：时分或时分秒，分隔符为":"。例如'YYYY/MM/DD'、'YYYYMMDD HH24:MI:SS'、'HH24:MI'，其中 YYYYMMDD、HH24MISS 为格式符；/:为分割符。

DM 缺省的日期语法格式 FMT 为：'YYYY-MM-DD HH:MI:SS.FF6'。

例

```
SQL> SELECT TO_date(' 20190215 14.47.38','YYYY-MM-DD HH24:MI:SS');
```

查询结果：2019-02-15 14:47:38

```
SQL> SELECT TO_date(' 20190215 14.47.38','YYYY-MM-DD
HH24:MI:SS','NLS_DATE_LANGUAGE=''AMERICAN''');
```

查询结果：2019-02-15 14:47:38.000000

日期语法格式 FMT 中的格式符、分隔符和 FX 固定格式器。详细介绍如下：

- **格式符**

日期语法格式 FMT 中的格式符由年、月、日、时、分、秒等元素组成。详细的元素介绍，参见表 8.3.4。

表 8.3.4 格式符

元素	说明	区别
D	周中的某一天，星期天算起	
DD	月中的某一天	
DDD	年中的某一天	
HH HH12 HH24	天中的时(0－23)。 HH，HH12 为 12 小时制。HH24 为 24 小时制	
MI	分(0－59)	
MM	月(01－12)	
SS	秒(0－59)	
SSSSS	一天从午夜开始的累积秒数(0-86399)	
TZH	时区中的小时，例如 hh:mi:ss.fftzh:tzh'	

TZM	时区中的分钟	
FF[1…9]	毫秒，[1…9]指定毫秒的精度，不指定时缺省为 9	
SSXFF	X 表示秒和毫秒的间隔，等价于.	
YYYY	4 位的年份	
YY	年份的最后 2 位数字	
Y	年份的最后 1 位数字	
AD/A.D.	公元,不能为 0	
AM/A.M.	上午	
BC/B.C.	公元前	
CC/SCC	世纪	不适用于 TO_DATE 中
DAY	星期（如星期五或FRIDAY）	
DL	返回长日期格式，包括年月日和星期几	
DS	返回短日期格式，包括年月日	
DY	星期的缩写形式（如星期五或 FRI）	
IW	星期数（当前日期所在星期是这一年的第几个星期，基于 ISO 标准）	不适用于 TO_DATE 中
MON	月份名称的缩写形式(如 12 月或 DEC)	
MONTH	月份名称（如 12 月或 DECEMBER）	
PM/P.M.	下午	
Q	季度号（1,2,3,4）	不适用于 TO_DATE 中
RR/RRRR	RR:输入参数年份的 2 位数字和数据库服务器上当前年的后 2 位数字(当年)共同确定 当指定的两位年份数字在 00~49 之间时：若当前年的后两位数字在 00~49 之间，则返回年份的前两位数字和当前年的前两位数字相同；若当前年的后两位数字在 50~99 之间，则返回年份的前两位数字为当前年的前两位数字加 1 当指定的两位年份数字在 50~99 之间时：若当前年份的后两位数字在 00~49 之间，则返回年份的前两位数字为当前年的前两位数字减 1；若当前年的后两位数字在 50~99 之间，则返回年份的前两位数字和当前年的前两位数字相同。 只有后面无其他分隔符且有其它格式符的情况才最多处理两位数字的年份。如:rrmm RRRR：如果输入参数只有两位，则同 RR，否则同 YYYY 作用	
WW	星期数（当前日期所在星期是这一年的第几个星期，第一个星期从 1 月 1 日开始，到 1 月 7 日结束）	
W	星期数（当前日期所在星期是这个月的第几个星期）	
Y,YYY	带逗号的年份，ISO 标准年份。	
IYYY,IYY,IY,I	最后倒数 4 位，3 位，2 位，1 位 ISO 标准年份。 ISO 标准认为日期是从周一到周日，按周计算。普通的标准则指定任何一年的一月一号都是周一	不适用于 TO_DATE 中
YYYY/SYYYY	ISO 标准年份，S 前缀表示公元前 BC	
YEAR/SYEAR	拼写出的年份（比如 TWENTY FIFTEEN）S 前缀表示负年	不适用于 TO_DATE 中

- **分隔符**

下面介绍 DATE 格式中用到的分隔符。分隔符分为两种：一是非限定分隔符，通常指除大小写字母、数字以及双引号之外的所有单字节字符且可打印的。例如：空格、回车键、tab 键、- / , . : *等标点符号。单个引号‟可以作为原串的分隔符，但是不能在 FMT 中作分割符。二是限定分隔符，指由双引号括起来的任意长度串，比如中文。例如‟年”‟月”‟日”里的年、月、日。

TO_DATE/TO_TIMESTAMP/TO_TIMESTAMP_TZ 函数目前支持的分隔符的规则如下：

1． 分隔符均包括尾空格，把尾空格全部当作分隔符，但不包括头空格，原串中对应分隔符处忽略头空格。分隔符没有尾空格，原串多出来的后面空格忽略。 tab 键与回车键也是如此。

```
SQL> select to_date('2001:  10:10','yyyy:mm:dd') from dual;
行号       TO_DATE('2001:10:10','yyyy:mm:dd')
---------------------------------------------------------------
1          2001-10-10 00:00:00
SQL> select to_date('2001--10:10','yyyy  mm:dd') from dual;
行号       TO_DATE('2001--10:10','yyyymm:dd')
---------------------------------------------------------------
1          2001-10-10 00:00:00
```

2． 原串中对应位置非限定分隔符的个数小于等于 FMT 中对应位置非限定分隔符的个数。

如果 FMT 设置了某个位置有连续 n 个（n>=1）非限定分隔符，非限定分隔符去除头空格后长度为 m（m>=1），则原串对应位置的非限定分隔符要小于等于 m 个。

例如，FMT 中第二个分隔符为 3 个连续的:，那么原串对应位置的分隔符要小于等于 3 个。

```
SQL> select to_date('2001-10--10','yyyy:mm:::dd') from dual;
行号       TO_DATE('2001-10--10','yyyy:mm:::dd')
---------------------------------------------------------
1          2001-10-10 00:00:00
```

3． 原串中对应位置限定分隔符的个数必须等于 FMT 中实际限定分隔符的个数。

如果 FMT 设置了连续 n 个（n>=1）限定分隔符， 限定分隔符去除头空格后长度为 m（m>=1），则源串对应位置必须有 m 个限定分隔符。

例如，FMT 中指定了"猪年"或"　猪年"作为实际限定分隔符，那么原串中也要指定相同的"猪年"。

```
SQL> select to_date('2019 猪年 10 月 10 日','yyyy"猪年"mm"月"dd"日"') from dual;
行号       TO_DATE('2019 猪年 10 月 10 日','yyyy"猪年"mm"月"dd"日"')
---------------------------------------------------------------
1          2019-10-10 00:00:00
或
SQL> select to_date('2019猪年10月10日','yyyy"  猪年"mm"月"dd"日"') from dual;
行号       TO_DATE('2019 猪年 10 月 10 日','yyyy"猪年"mm"月"dd"日"')
---------------------------------------------------------------
1          2019-10-10 00:00:00
```

4． 原串中相应位置（限定+非限定）分隔符的个数要等于 FMT 中相应位置限定分隔符的个数。

如果 FMT 某个位置设置了连续 n 个（n 大于等于 1）非限定分隔符+限定分割符， 限定分隔符去除头空格后长度为 m（m 大于等于 1），则源串对应位置必须有 m 个非限定分隔

符。(限定+非限定)分隔符组合中，非限定符不允许改变。

例如，-3 月和"-mm"月，-29 日和-dd"日"个数，限定符都需满足严格匹配。

```
SQL> select to_date('2019年-3月-29日','yyyy"年"-mm"月"-dd"日"') from dual;
行号       TO_DATE('2019年-3月-29日','yyyy"年"-mm"月"-dd"日"')
------------------------------------------------------------
1          2019-03-29 00:00:00
```

5. 如果 FMT 中只包含非限定分隔符，则原串中对应位置可以有与分隔符内容不相同的分隔符匹配。

如果 FMT 中只包含限定分隔符，则源串中对应位置必须有与实际分隔符内容相同的串匹配。

如果 FMT 中既包含限定分隔符，又包含非限定分隔符(不分顺序)，则原串中对应位置必须有与实际分隔符内容相同的串匹配。

```
SQL> select to_date('2001:1010','yyyy-mmdd') from dual;
行号       TO_DATE('2001:1010','yyyy-mmdd')
---------- -----------------------------------------------
1          2001-10-10 00:00:00
```

6. 如果 FMT 未设置分隔符，则源串对应位置不能有除空格外的分隔符，如果 FMT 中只有空格，则源串对应位置可以有空格，也可以没有。

```
SQL> select to_date('200112   10','yyyy  mmdd') from dual;
行号       TO_DATE('20011210','yyyymmdd')
----------------------------------------------------
1          2001-12-10 00:00:00
SQL> select to_date('200112    10','yyyymmdd') from dual;
行号       TO_DATE('20011210','yyyymmdd')
------------------------------------------------------
1          2001-12-10 00:00:00
```

7. 对于 TO_DATE/TO_TIMESTAMP/TO_TIMESTAMP_TZ 来说，如果 FMT 格式符 XFF 前同时出现非限定分隔符.，不论有多少个.，分隔符.都会被忽略，只都相当于一个 XFF。

例如，在 TO_TIMESTAMP 中...XFF 相当于 XFF。

```
SQL> SELECT TO_TIMESTAMP ('10秒.123000', 'SS"秒"...XFF') FROM DUAL;
行号       TO_TIMESTAMP('10秒.123000','SS"秒"...XFF')
-------------------------------------------------------------
1          2019-01-01 00:00:10.123000
SQL> SELECT TO_TIMESTAMP ('10.123000', 'SS.XFF') FROM DUAL;
行号       TO_TIMESTAMP('10.123000','SS.XFF')
-------------------------------------------------------------
2019-01-01 00:00:10.123000
```

8. 源串中的数据不能多于 FMT 中格式符指定的位数。结尾可以少于 FMT 对应的位数。

- **FX 固定格式器**

FX 是 FMT 固定格式全局修改器。使用了 FX 之后，要求源串对应位置的内容必须和 FMT

中 FX 之后的格式严格匹配。FX 可以出现在任何分隔符可以出现的位置。

FX 专门应用于含有限定分隔符的或 fx 标记的 FMT 中。只有全是非限定分隔符的 FMT 或者属于快速格式的 FMT 中，FX 不起作用。快速格式的 FMT 共 12 种，分别为：YYYY-MM-DD (YYYY/MM/DD)、YYYY-DD-MM (YYYY/DD/MM)、MM-DD-YYYY (MM/DD/YYYY)、MM-YYYY-DD (MM/YYYY/DD)、DD-MM-YYYY (DD/MM/YYYY)、DD-YYYY-MM (DD/YYYY/MM)、HH:MI:SS、SS:MI:HH、YYYY-MM-DD HH:MI:SS (YYYY/MM/DD HH:MI:SS)、YYYY-MM-DD HH:MI:SS.ff[n] (YYYY/MM/DD HH:MI:SS[n])、YYYYMMDD、YYYYMMDD HH:MI:SS。

例

无 FX 情况下，源串格式和 FMT 不需要完全匹配（非限定分隔符个数少于等于源串、非限定分隔符内容不同，固定格式位数不一样等模糊匹配），也能执行成功。

```
SQL> SELECT TO_DATE('19 年 08 月 01','yyyy"年"mm"月"dd') FROM DUAL;
行号       TO_DATE('19 年 08 月 01','yyyy"年"mm"月"dd')
----------------------------------------------------------------------------
1          19-08-01 00:00:00
```

有 FX 情况下，源串格式和 FMT 没有完全匹配，报错：文字与格式字符串不匹配。

```
SQL> SELECT TO_DATE('19 年 08 月 01','fxyyyy"年"mm"月"dd') FROM DUAL;
SELECT TO_DATE('19 年 08 月 01', 'fxyyyy"年"mm"月"dd') FROM DUAL;
[-6130]:文字与格式字符串不匹配.
```

有 FX 情况下，FX 位于 FMT 最前端，此时源串格式需要和 FMT 完全匹配，才能执行成功。

```
SQL> SELECT TO_DATE('2019 年 08 月 01','fxyyyy"年"mm"月"dd') FROM DUAL;
行号       TO_DATE('2019 年 08 月 01','fxyyyy"年"mm"月"dd')
----------------------------------------------------------------------------
1          2019-08-01 00:00:00
```

38．函数 FROM_TZ

```
    语法：FROM_TZ(timestamp,timezone|tz_name])
```

功能：将时间戳类型 timestamp 和时区类型 timezone（或时区名称 tz_name）转化为 timestamp with timezone 类型 。

timestamp 缺省的日期语法为："YYYYMMDD HH:MI:SS"或者"YYYYMMDD "。

时区设置范围为：-12:59~+14:00。

时区名：ASIA/HONG_KONG（即+08:00）。

例 使用时区：

```
    SELECT   FROM_TZ(TO_TIMESTAMP('20091101   09:10:21','YYYYMMDD   HH:MI:SS
'),'+09:00') ;
```

查询结果：2009-11-01 09:10:21.000000 +09:00

使用时区名：

```
    SELECT FROM_TZ(TO_TIMESTAMP('20091101','YYYYMMDD'),'ASIA/HONG_KONG') ;
```

查询结果：2009-11-01 00:00:00.000000 +08:00

不指定格式：

```
    select from_tz('20091101', 'ASIA/HONG_KONG');
```

查询结果：2009-11-01 00:00:00.000000 +08:00

39．函数 TRUNC

```
语法：TRUNC(date[, fmt])
```

功能：将日期时间 date 截断到最接近格式参数 fmt 指定的形式。若 fmt 缺省，则返回当天日期。语法与 ROUND 类似，但结果是直接截断，而不是四舍五入。参数及函数的返回类型与 ROUND 相同。参见 ROUND。

例

```
SELECT TRUNC(DATE '1992-10-27', 'scc');
```

查询结果为：1901-01-01

```
SELECT TRUNC(DATE '1992-10-27', 'YEAR') "FIRST OF THE YEAR";
```

查询结果为：1992-01-01

```
SELECT TRUNC(DATE '1992-10-27', 'q');
```

查询结果为：1992-10-01

```
SELECT TRUNC(DATE '1992-10-27', 'month');
```

查询结果为：1992-10-01

```
SELECT TRUNC(TIMESTAMP '1992-10-27 11:00:00', 'ww');
```

查询结果为：1992-10-21 00:00:00.000000

```
SELECT TRUNC(TIMESTAMP '1992-10-27 11:00:00', 'w');
```

查询结果为：1992-10-22 00:00:00.000000

```
SELECT TRUNC(TIMESTAMP '1992-10-27 12:00:01', 'ddd');
```

查询结果为：1992-10-27 00:00:00.000000

```
SELECT TRUNC(DATE '1992-10-27', 'day');
```

查询结果为：1992-10-25

```
SELECT TRUNC(TIMESTAMP '1992-10-27 12:00:31', 'hh');
```

查询结果为：1992-10-27 12:00:00.000000

```
SELECT TRUNC(TIMESTAMP '1992-10-27 12:00:31', 'mi');
```

查询结果为：1992-10-27 12:00:00.000000

40．函数 WEEK

```
语法：WEEK(date)
```

功能：返回指定日期属于所在年中的第几周。

例

```
SELECT WEEK(DATE '2003-02-10');
```

查询结果为：7

41．函数 WEEKDAY

```
语法：WEEKDAY(date)
```

功能：返回指定日期的星期值。如果是星期日则返回 0。

例

```
SELECT WEEKDAY(DATE '1998-10-26');
```

查询结果：1

42．函数 WEEKS_BETWEEN

```
语法：WEEKS_BETWEEN(date1,date2)
```

功能：返回两个日期之间相差周数。

例

```
SELECT WEEKS_BETWEEN(DATE '1998-2-28', DATE '1998-10-31');
```

查询结果：-35

43. 函数 YEAR

```
语法：YEAR(date)
```

功能：返回日期中的年分量。

例

```
SELECT YEAR(DATE '2001-05-12');
```

查询结果为：2001

44. 函数 YEARS_BETWEEN

```
语法：YEARS_BETWEEN(date1,date2)
```

功能：返回两个日期之间相差年数。

例

```
SELECT  YEARS_BETWEEN(DATE '1998-2-28', DATE '1999-10-31');
```

查询结果为： -1

45. 函数 LOCALTIME

```
语法：LOCALTIME (n)
```

功能：返回当前时间值，结果类型为 TIME。等价于 CURTIME()。

参数：n：指定毫秒的精度。取值范围 0-6，默认为 6。

46. 函数 LOCALTIMESTAMP

```
语法：LOCALTIMESTAMP (n)
```

功能：返回当前日期时间值，结果类型为 TIMESTAMP。

参数：n：指定毫秒的精度。取值范围 0-6，默认为 6。

47. 函数 OVERLAPS

```
语法：OVERLAPS (date1,date2,date3,date4)
```

功能：返回两个时间段是否存在重叠，date1 为 datetime 类型、date2 可以为 datetime 类型也可以为 interval 类型，date3 为 datetime 类型，date4 可为 datetime 类型，也可以 interval 类型，判断(date1,date2),(date3,date4)有无重叠。其中 date2 与 date4 类型必须一致，如果 date2 为 interval year to month，date4 也必须是此类型。结果类型为 BIT，若两个时间段存在重叠返回 1，不重叠返回 0。

例

```
SELECT OVERLAPS('2011-10-3','2011-10-9','2011-10-6','2011-10-13');
```

查询结果为：1

```
SELECT OVERLAPS('2011-10-3','2011-10-9','2011-10-10','2011-10-11');
```

查询结果为：0

```
SELECT OVERLAPS('2011-10-3',INTERVAL '09 23' DAY TO HOUR,'2011-10-10',INTERVAL '09 23' DAY TO HOUR);
```

查询结果为：1

```
SELECT OVERLAPS('2011-10-3',INTERVAL '01 23' DAY TO HOUR,'2011-10-10',INTERVAL '09 23' DAY TO HOUR);
```

查询结果为：0

```
SELECT OVERLAPS('2011-10-3',INTERVAL '1' YEAR TO
```

```
MONTH,'2012-10-10',INTERVAL '1-1' YEAR TO MONTH);
```

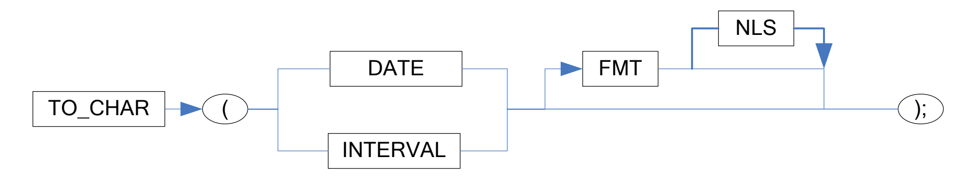
查询结果为：0

```
    SELECT      OVERLAPS('2011-10-3',INTERVAL     '2'     YEAR     TO
MONTH,'2012-10-10',INTERVAL '1-1' YEAR TO MONTH);
```

查询结果为：1

48. 函数 TO_CHAR

```
语法：TO_CHAR(date[,fmt[,nls]])
```



功能：将日期数据类型 DATE 转换为一个在日期语法格式(FMT)中指定语法的 VARCHAR 类型字符串。若没有指定语法，日期 DATE 将按照缺省的语法转换为一个 VARCHAR 值。

FMT，NLS 的用法请参考函数 TO_DATE/TO_TIMESTAMP/TO_TIMESTAMP_TZ 用法。

DM 缺省的日期语法格式 FMT 为：'YYYY-MM-DD HH:MI:SS.FF6'。

例

```
SELECT TO_CHAR(SYSDATE,'YYYYMMDD');
```

查询结果：20110321 /* SYSDATE 为系统当前时间*/

```
SELECT TO_CHAR(SYSDATE,'YYYY/MM/DD');
```

查询结果：2011/03/21

```
SELECT TO_CHAR(SYSDATE,'HH24:MI');
```

查询结果：16:56

```
SELECT TO_CHAR(SYSDATE,'YYYYMMDD HH24:MI:SS');
```

查询结果：20110321 16:56:19

```
SELECT TO_CHAR(SYSDATE,'YYYY-MM-DD HH24:MI:SS');
```

查询结果：2011-03-21 16:56:28

```
SELECT TO_CHAR(INTERVAL '123-2' YEAR(3) TO MONTH) FROM DUAL;
```

查询结果：INTERVAL '123-2' YEAR(3) TO MONTH

```
select to_char(sysdate(), 'mon', 'NLS_DATE_LANGUAGE = ENGLISH');
```

查询结果：DEC

```
    select to_char(sysdate(),'mon','NLS_DATE_LANGUAGE=''SIMPLIFIED CHINESE''
');
```

查询结果：12 月

49. 函数 SYSTIMESTAMP

```
语法：SYSTIMESTAMP (n)
```

功能：返回系统当前的时间戳，带数据库的时区信息。结果类型为 TIMESTAMP WITH TIME ZONE。

参数：n：指定毫秒的精度。取值范围 0-6，默认为 6

例

```
SELECT SYSTIMESTAMP();
```

查询结果为：2012-10-10 11:06:12.171000 +08:00

50．函数 NUMTODSINTERVAL

```
语法：NUMTODSINTERVAL (number, interval_unit)
```

功能：转换一个指定的 DEC 类型到 INTERVAL DAY TO SECOND

参数：

number：任何 dec 类型的值或者可以转换到 dec 类型的表达式

interval_unit：字符串限定 number 的类型： DAY、HOUR、MINUTE、或 SECOND

例

```
SELECT NUMTODSINTERVAL (2.5,'DAY');
```

查询结果为：INTERVAL '000000002 12:00:00.000000' DAY(9) TO SECOND(6)

51．函数 NUMTOYMINTERVAL

```
语法：NUMTOYMINTERVAL (number, interval_unit)
```

功能：转换一个指定的 DEC 类型值到 INTERVAL YEAR TO MONTH

参数：

number：任何 dec 类型的值或者可以转换到 dec 类型的表达式

interval_unit：字符串限定 number 的类型：YEAR 或 MONTH

例

```
SELECT NUMTOYMINTERVAL (2.5,'YEAR');
```

查询结果为：INTERVAL '000000002-06' YEAR(9) TO MONTH

52．函数 WEEK

```
语法：WEEK(date, mode)
```

功能：根据指定的 mode 返回日期为所在年的第几周

其中 mode 可取值及其含义见下表。

表 8.3.5 mode 取值及其含义

mode 值	周起始	返回值范围	说明
0	周日	0-53	本年第一个周日开始为第 1 周，之前算本年第 0 周
1	周一	0-53	本年第一个周一之前如果超过 3 天则算第 1 周，否则算第 0 周
2	周日	1-53	本年第一个周日开始为第 1 周，之前算去年第 5x 周
3	周一	1-53	本年第一个周一之前如果超过 3 天则算第 1 周，否则算去年第 5x 周；年末不足 4 天算明年第 1 周
4	周日	0-53	本年第一个周日之前如果超过 3 天则算第 1 周，否则算第 0 周
5	周一	0-53	本年第一个周一开始为第 1 周，之前算本年第 0 周
6	周日	1-53	本年第一个周日之前如果超过 3 天算第 1 周，否则算去年第 5x 周；年末不足 4 天算明年第 1 周
7	周一	1-53	本年第一个周一开始为第 1 周，之前算去年第 5x 周

mode 的取值范围为-2147483648~2147483647，但在系统处理时会取 mode= mode%8。

由于 DM 支持儒略历，并考虑了历史上从儒略历转换至格里高利日期时的异常，不计算'1582-10-05'到'1582-10-14'之间的 10 天，因此 WEEK 函数对于 1582-10-15 之前日期的计算结果不能保证正确性。

例

```
SELECT WEEK('2013-12-31',0);
```

查询结果为：52

```
SELECT WEEK('2013-12-31',1);
```

查询结果为：53

```
SELECT WEEK('2013-12-31',2);
```

查询结果为：52

```
SELECT WEEK('2013-12-31',3);
```

查询结果为：51

53. 函数 UNIX_TIMESTAMP

```
语法：UNIX_TIMESTAMP (d datetime)
```

功能：自标准时区的'1970-01-01 00:00:00 +0:00'到本地会话时区的指定时间的秒数差。如果为空，表示到当前时间。

参数：d 可以是一个 DATETIME、TIME、DATE、timestamp with time zone、timestamp with LOCAL time zone 类型（时区忽略，使用当前时区），也可以是一个字符串。或一个 YYYYMMDD、YYMMDD、YMMDD、YYYMMDD 格式的整形 BIGINT。

例

当前会话时区是+8:00

```
SELECT UNIX_TIMESTAMP(timestamp '1970-01-01 08:00:00');
```

查询结果：0

```
SELECT UNIX_TIMESTAMP('1970-01-01 17:00:00');
```

查询结果：28800

```
SELECT UNIX_TIMESTAMP( 20120608 );
```

查询结果：1339084800

54. 函数 FROM_UNIXTIME

```
语法 1：FROM_UNIXTIME (unixtime int)（bigint 返回 null）
```

功能：将自'1970-01-01 00:00:00'的秒数差转成本地会话时区的时间戳类型。

unixtime 为需要处理的参数(该参数是 Unix 时间戳)，可以直接是 Unix 时间戳字符串。

例

```
select FROM_UNIXTIME ('539712061');
```

查询结果：1987-02-08 01:01:01

```
SELECT FROM_UNIXTIME(1249488000)
```

查询结果：2009-08-06 01:00:00

```
语法 2：FROM_UNIXTIME (unixtime int,fmt varchar)
```

功能：将自'1970-01-01 00:00:00'的秒数差转成本地会话时区的指定 fmt 格式的时间串。

unixtime 为需要处理的参数(该参数是 Unix 时间戳)，

可以直接是 Unix 时间戳字符串(不支持).

Fmt 见 **DATE_FORMAT** 中的 **format** 格式。

例

```
SELECT FROM_UNIXTIME( 1249488000 ,'%D')
```

查询结果：6th

55. 函数 SESSIONTIMEZONE

```
语法：SESSIONTIMEZONE
```

功能：查看当前会话的时区

例

```
SELECT SESSIONTIMEZONE FROM DUAL;
```

查询结果：

```
行号        SESSIONTIMEZONE
---------- ---------------
1          +08:00
```

56. 函数 DATE_FORMAT

```
语法：DATE_FORMAT (d  datetime, format varchar)
```

功能：以不同的格式显示日期/时间数据。

参数：

d：可以是可以是一个 DATETIME、TIME、DATE、timestamp with time zone、timestamp with LOCAL time zone 类型（时区忽略，使用当前时区）、一个 YYYYMMDD YYMMDD YMMDD YYYYMMDD 格式的整形（不支持）。

format： 规定日期/时间的输出格式。格式有以下：27 个（NG 表示暂不支持）。非格式不做解释直接照抄。

表 8.3.1 format 释义

format	释义	备注
%a	缩写星期名	
%b	缩写月名	
%c	月，数值(0-12)	暂不支持
%D	带有英文前缀的月中的天	
%d	月的天，数值(00-31)	
%e	月的天，数值(0-31)	暂不支持
%f	微秒	
%H	小时 (00-23)	
%h	小时 (01-12)	
%I	小时 (01-12)	
%i	分钟，数值(00-59)	
%j	年的天 (001-366)	
%k	小时 (0-23)	暂不支持
%l	小时 (1-12)	暂不支持
%M	月名	
%m	月，数值(00-12)	
%p	AM 或 PM	
%r	时间，12-小时（hh:mm:ss AM 或 PM）	
%S	秒(00-59)	
%s	秒(00-59)	
%T	时间, 24-小时 (hh:mm:ss)	

%U	周 (00-53) 星期日是一周的第一天	
%u	星期(0……52), 这里星期一是星期的第一天	暂不支持
%W	星期名字（Sunday、Tuesday、Wednesday、Thursday、Friday、Saturday）	
%Y	年, 数字, 4 位	
%y	年, 数字, 2 位	
%%	一个文字“%”	

例

```
select date_format(timestamp '1980-1-1 1:1:1.123456789','%Y-%m-%d %H:%i:%s');
```

查询结果：1980-01-01 01:01:01

57. 函数 TIME_TO_SEC

```
语法：TIME_TO_SEC (d datetime)
```

功能：将时间换算成秒

d 可以是一个 DATETIME、TIME、DATE、timestamp with time zone、timestamp with LOCAL time zone 类型（时区忽略，使用当前时区），

例

```
select time_to_sec(timestamp '1900-1-1 23:59:59 +8:00');
```

查询结果：86399

```
select time_to_sec(time '23:59:59');
```

查询结果：86399

58. 函数 SEC_TO_TIME

```
语法：SEC_TO_TIME (sec int)
```

功能：将秒换算成时间

例

```
select sec_to_time(104399);
```

查询结果： 28:59:59

59. 函数 TO_DAYS

```
语法：TO_DAYS (d timestamp)
```

功能：转换成公元 0 年 1 月 1 日的天数差。

d 是要转换的日期时间类型。或一个 YYYYMMDD YYMMDD YMMDD YYYMMDD 格式的整形 BIGINT。

例

```
SQL> select  to_days(now());
```

查询结果： 737614

8.4 空值判断函数

空值判断函数用于判断参数是否为 NULL，或根据参数返回 NULL。

1. 函数 COALESCE

```
语法：COALESCE(n1,n2,…,nx)
```

功能：返回其参数中第一个非空的值，如果所有参数均为 NULL，则返回 NULL。如果参数为多媒体数据类型，如 TEXT 类型，则系统会将 TEXT 类型先转换为 VARCHAR 类型或 VARBINARY 类型，转换的最大长度为 8188，超过部分将被截断。

例

```
SELECT COALESCE(1,NULL);
```

查询结果：1

```
SELECT COALESCE(NULL,TIME '12:00:00',TIME '11:00:00');
```

查询结果：12:00:00

```
SELECT COALESCE(NULL,NULL,NULL,NULL);
```

查询结果：NULL

2. 函数 IFNULL

```
语法：IFNULL(n1,n2)
```

功能：当表达式 n1 为非 NULL 时，返回 n1；若 n1 为 NULL，则返回表达式 n2 的值。若 n1 与 n2 为不同数据类型时，DM 会进行隐式数据类型转换，若数据类型转换出错，则会报错。

例

```
SELECT IFNULL(1,3);
```

查询结果：1

```
SELECT IFNULL(NULL,3);
```

查询结果：3

```
SELECT IFNULL('',2);
```

查询结果：（空串）

3. 函数 ISNULL

```
语法：ISNULL(n1,n2)
```

功能：当表达式 n1 为非空时，返回 n1；若 n1 为空，则返回表达式 n2 的值。n2 的数据类型应能转为 n1 的数据类型，否则会报错。

例

```
SELECT ISNULL(1,3);
```

查询结果：1

4. 函数 NULLIF

```
语法：NULLIF(n1,n2)
```

功能：如果 n1=n2，返回 NULL，否则返回 n1。

例

```
SELECT NULLIF(1,2);
```

查询结果：1

```
SELECT NULLIF(1,1);
```

查询结果：NULL

5. 函数 NVL

```
语法：NVL(n1,n2)
```

功能：返回第一个非空的值。若 n1 与 n2 为不同数据类型时，DM 会进行隐式数据类型

转换，若数据类型转换出错，则会报错。

说明：

1） 当 n1 为确定性数据类型时，以 n1 为准；当 n1 的数据类型不确定时，以找到的第一个确定性数据类型为准；如果都为不确定数据类型时则定为 varchar 数据类型；
2） 若参数一个为精确浮点数另一个为不精确浮点数，结果为不精确浮点数，可能导致数据精度丢失；
3） 两个参数为不同数据类型时，结果为精度大的数据类型；
4） 参数若为字符串类型，不论是 char 还是 varchar，结果类型均为 varchar，精度以大的为准；
5） 参数类型都为时间日期类型时，结果类型为 n1 的类型；但若参数有 DATE 或 TIME 类型时，结果类型为 n1 和 n2 中精度较大的类型。

6．函数 NULL_EQU

```
语法：NULL_EQU(n1,n2)
```

功能：返回两个类型相同的值的比较，当 n1=n2 或 n1、n2 两个值中出现 null 时，返回 1。类型可以是 INT、BIT、BIGINT、FLOAT、DOUBLE、DEC、VARCHAR、DATE、TIME、TIME ZONE、DATETIME、DATETIME ZONE、INTERVAL 等。

例

```
select null_equ(1,1);
```

查询结果为：1

```
select null_equ(1,3);
```

查询结果为：0

```
select null_equ(1,null);
```

查询结果为：1

8.5 类型转换函数

1．函数 CAST

```
语法：CAST(value AS type)
```

功能：将参数 value 转换为 type 类型返回。类型之间转换的相容性如下表所示：表中，"允许"表示这种语法有效且不受限制，"—"表示语法无效，"受限"表示转换还受到具体参数值的影响。

数值类型为：精确数值类型和近似数值类型。

精确数值类型为：NUMERIC、DECIMAL、BYTE、INTEGER、SMALLINT。

近似数值类型为：FLOAT、REAL、DOUBLE PRECISION。

字符串为：变长字符串和固定字符串。

变长字符串为：VARCHAR、VARCHAR2。

固定字符串为：CHAR、CHARACTER。

字符串大对象为：CLOB、TEXT。

二进制为：BINARY、VARBINARY。

二进制大对象为：BLOB、IMAGE。

日期为：DATE。时间为：TIME。时间戳为：TIMESTAMP。

时间时区为：TIME WITH TIME ZONE。

时间戳时区为：TIMESTAMP WITH TIME ZONE。

年月时间间隔为：INTERVAL YEAR TO MONTH、INTERVAL YEAR、INTERVAL MONTH。

日时时间间隔为：INTERVAL DAY、INTERVAL DAY TO HOUR、INTERVAL DAY TO MINUTE、INTERVAL DAY TO SECOND、INTERVAL HOUR、INTERVAL HOUR TO MINUTE、INTERVAL HOUR TO SECOND、INTERVAL MINUTE、INTERVAL MINUTE TO SECOND、INTERVAL SECOND。

表 8.5.1 CAST 类型转换相容矩阵

Value	**type 数据类型**											
数据类型	数值类型	字符串	字符串大对象	二进制	二进制大对象	日期	时间	时间戳	时间时区	时间戳时区	年月时间间隔	日时时间间隔
数值类型	受限	受限	—	允许	—	受限	受限	受限	—	—	受限	受限
字符串	允许	允许	允许	允许	允许	受限	受限	受限	受限	受限	允许	允许
字符串大对象	—	允许	—	—	允许	—	—	—	—	—	—	—
二进制	允许	允许	—	允许	允许	—	—	—	—	—	—	—
二进制大对象	—	—	—	—	允许	—	—	—	—	—	—	—
日期	—	允许	—	—	—	允许	—	允许	—	允许	—	—
时间	—	允许	—	—	—	—	允许	允许	允许	允许	—	—
时间戳	—	允许	—	—	—	允许	允许	允许	允许	允许	—	—
时间时区	—	允许	—	—	—	—	允许	允许	允许	—	—	—
时间戳时区	—	允许	—	—	—	允许	允许	允许	—	允许	—	—
年月时间间隔	—	允许	—	—	—	—	—	—	—	—	允许	—
日时时间间隔	—	允许	—	—	—	—	—	—	—	—	—	允许

例

```
SELECT CAST(100.5678 AS NUMERIC(10,2));
```

查询结果：100.57

```
SELECT CAST(100.5678 AS VARCHAR(8));
```

查询结果：100.5678

```
SELECT CAST('100.5678' AS INTEGER);
```

查询结果：101

```
SELECT CAST(INTERVAL '01-01' YEAR TO MONTH AS char(50));
```

查询结果：INTERVAL '1-1' YEAR(9) TO MONTH

2．函数 CONVERT

```
语法：CONVERT(type,value)
```

功能：将参数 value 转换为 type 类型返回。其类型转换相容矩阵与函数 CAST()的相同。

例

```
SELECT CONVERT(VARCHAR(8),100.5678);
```

查询结果：100.5678

```
SELECT CONVERT(INTEGER, '100.5678');
```

查询结果：101

```
SELECT CONVERT(CHAR(50),INTERVAL '100-5' YEAR(3) TO MONTH);
```

查询结果：INTERVAL '100-5' YEAR(3) TO MONTH

3. 函数 HEXTORAW

```
语法：HEXTORAW (string)
```

功能：将由 string 表示的二进制字符串转换为一个 binary 数值类型。

例

```
SELECT HEXTORAW ('abcdef');
```

查询结果为：0xABCDEF

```
SELECT HEXTORAW ('B4EFC3CECAFDBEDDBFE2D3D0CFDEB9ABCBBE');
```

查询结果为：0xB4EFC3CECAFDBEDDBFE2D3D0CFDEB9ABCBBE

4. 函数 RAWTOHEX

```
语法：RAWTOHEX (binary)
```

功能：将 RAW 类数值 binary 转换为一个相应的十六进制表示的字符串。binary 中的每个字节都被转换为一个双字节的字符串。 RAWTOHEX 和 HEXTORAW 是两个相反的函数。

例

```
SELECT RAWTOHEX('达梦数据库有限公司');
```

查询结果为：B4EFC3CECAFDBEDDBFE2D3D0CFDEB9ABCBBE

```
SELECT RAWTOHEX('13');
```

查询结果为：3133

5. 函数 BINTOCHAR

```
语法：BINTOCHAR (binary)
```

功能：将数值 binary 转换为字符串。

例

```
SELECT BINTOCHAR ('0x61626364');
```

查询结果为：abcd

6.函数 TO_BLOB

```
语法：TO_BLOB (varbinary)
```

功能：将数值 varbinary 转换为 blob。

例

```
SELECT TO_BLOB(utl_raw.cast_to_raw('abcd'));
```

查询结果为：0x61626364

8.6 杂类函数

1. 函数 DECODE

```
语法：DECODE(exp, search1, result1, … searchn, resultn[,default])
```

功能：查表译码，DECODE 函数将 exp 与 search1,search2, … searchn 相比较，如果等于 searchx，则返回 resultx，如果没有找到匹配项，则返回 default，如果未定义 default，返回 NULL。

例

```
SELECT DECODE(1, 1, 'A', 2, 'B');
```

查询结果为：'A'

```
SELECT DECODE(3, 1, 'A', 2, 'B');
```

查询结果为：NULL

```
SELECT DECODE(3, 1, 'A', 2, 'B', 'C');
```

查询结果为：'C'

2． 函数 ISDATE

```
语法：ISDATE(exp)
```

功能：判断给定表达式是否为有效的日期，是返回 1，否则返回 0。

例

```
SELECT ISDATE('2012-10-9');
```

查询结果为：1

```
SELECT ISDATE('2012-10-9 13:23:37');
```

查询结果为：1

```
SELECT ISDATE(100);
```

查询结果为：0

3． 函数 ISNUMERIC

```
语法：ISNUMERIC(exp)
```

功能：判断给定表达式是否为有效的数值，是返回 1，否则返回 0。

例

```
SELECT ISNUMERIC(1.323E+100);
```

查询结果为：1

```
SELECT ISNUMERIC('2a');
```

查询结果为：0

4． 函数 DM_HASH

```
语法：DM_HASH (exp)
```

功能：根据给定表达式生成 HASH 值，返回结果为整型。

例

```
SELECT DM_HASH('DM HASH VALUE');
```

查询结果为：3086393668

```
SELECT DM_HASH(101);
```

查询结果为：1653893674

5． 函数 LNNVL

```
语法：LNNVL(condition)
```

参数：condition为布尔表达式。

功能：如果condition表达式计算结果值为FALSE或者UNKNOWN时，返回TRUE，计算为TRUE时，则返回FALSE。

例

```
SELECT *FROM DUAL WHERE LNNVL(1=0) ;
```

查询结果：1

```
SELECT T1.NAME, T2.NAME
FROM  PRODUCTION.PRODUCT_CATEGORY T1  RIGHT OUTER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID WHERE LNNVL(T1.NAME<>'计算机');
```

查询结果：

NAME	NAME
null	历史
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体

6. 函数 LENGTHB

```
语法：LENGTHB(value)
```

参数：value为CHAR/VARCHAR/BINARY/VARBINARY/BLOB/CLOB类型的值。

功能：返回value的字节数。

例

```
SELECT LENGTHB(0x1234567) "Length in bytes";
```

查询结果为：4

7. 函数 FIELD

```
语法：函数 FIELD(value, e1, e2, e3, e4...en)
```

功能：根据指定元素value在输入列表“e1、e2、e3、e4...en”中的位置返回相应的位置序号，不在输入列表时则返回0。

FIELD()一般用在ORDER BY子句之后，将获取到的结果集按照输入列表的顺序进行排序。value不在输入列表的结果，排在结果集的前面。

例

1）查询50在后面列表10、50、100中的位置序号。

```
SELECT field(50,10,50,100);
```

查询结果为：2

2）按照列表中指定的顺序输出结果集。不符合条件的结果放在结果集前面。

```
select * from PERSON.ADDRESS order by field(city,'武汉市洪山区','武汉市汉阳区','武汉市武昌区','武汉市江汉区');
```

查询结果为：

```
ADDRESSID   ADDRESS1     ADDRESS2     CITY         POSTALCODE
3    青山区青翠苑 1 号                      武汉市青山区      430080
6    洪山区保利花园 50-1-304                武汉市洪山区      430073
1    洪山区 369 号金地太阳城 56-1-202       武汉市洪山区      430073
16   洪山区光谷软件园 C1_501                武汉市洪山区      430073
13   洪山区关山春晓 55-1-202                武汉市洪山区      430073
15   洪山区关山春晓 11-1-202                武汉市洪山区      430073
7    洪山区保利花园 51-1-702                武汉市洪山区      430073
8    洪山区关山春晓 51-1-702                武汉市洪山区      430073
2    洪山区 369 号金地太阳城 57-2-302       武汉市洪山区      430073
14   洪山区关山春晓 10-1-202                武汉市洪山区      430073
5    汉阳大道熊家湾 15 号                   武汉市汉阳区      430050
11   武昌区武船新村 1 号                    武汉市武昌区      430063
4    武昌区武船新村 115 号                  武汉市武昌区      430063
12   江汉区发展大道 423 号                  武汉市江汉区      430023
10   江汉区发展大道 555 号                  武汉市江汉区      430023
9    江汉区发展大道 561 号                  武汉市江汉区      430023
```

第 9 章 一致性和并发性

数据一致性是指表示客观世界同一事物状态的数据，不管出现在何时何处都是一致的、正确的和完整的。数据库是一个共享资源，可为多个应用程序所共享，它们同时存取数据库中的数据，这就是数据库的并发操作。此时，如果不对并发操作进行控制，则会存取不正确的数据，或破坏数据库数据的一致性。

DM 利用事务和封锁机制提供数据并发存取和数据完整性。在一事务内由语句获取的全部封锁在事务期间被保持，防止其它并行事务的破坏性干扰。一个事务的 SQL 语句所做的修改在它提交后才可能为其它事务所见。

DM 自动维护数据库的一致性和完整性，并允许选择实施事务级读一致性，它保证同一事务内的可重复读，为此 DM 提供用户各种手动上锁语句和设置事务隔离级别语句。

本章介绍 DM 中和事务管理相关的 SQL 语句和手动上锁语句。在本章各例中，如不特别说明，各例的当前用户均为建表者 SYSDBA。

9.1 DM 事务相关语句

DM 中事务是一个逻辑工作单元，由一系列 SQL 语句组成。DM 把一个事务的所有 SQL 语句作为一个整体，即事务中的操作，要么全部执行，要么一个也不执行。

9.1.1 事务的开始

DM 没有提供显式定义事务开始的语句，第一个可执行的 SQL 语句(除登录语句外)隐含事务的开始。

9.1.2 事务的结束

用户可以使用显式的提交或回滚语句来结束一个事务，也可以隐式地提交一个事务。

1．提交语句

语法格式

```
COMMIT [WORK];
```

参数

WORK 支持与标准 SQL 的兼容性，COMMIT 和 COMMIT WORK 等价。

功能

该语句使当前事务工作单元中的所有操作“永久化”，并结束该事务。

举例说明

例 1 插入数据到表 DEPARTMENT 并提交。

```
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('采购部门');
COMMIT WORK;
```

2．回滚语句

语法格式

```
ROLLBACK [WORK];
```

功能

该语句回滚(废除)当前事务工作单元中的所有操作，并结束该事务。

使用说明

建议用户退出时，用 COMMIT 或 ROLLBACK 命令来显式地结束应用程序。如果没有显式地提交事务，而应用程序又非正常终止，则最后一个未提交的工作单元被回滚。特别说明：CREATE TABLESPACE 和 ALTER DATABASE 两种 DDL 语句是不能回滚的。

举例说明

例 插入数据到表 DEPARTMENT 后回滚。

```
(1)往表 DEPARTMENT 中插入一个数据
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('销售部门');
(2)查询表 DEPARTMENT
SELECT * FROM RESOURCES.DEPARTMENT;
/*部门名为'销售部门'的记录可查询到*/
(3)回滚插入操作
ROLLBACK WORK;
(4)查询表 DEPARTMENT
SELECT * FROM RESOURCES.DEPARTMENT;
/*插入操作被回滚，表 DEPARTMENT 中不存在部门名为'销售部门'的记录*/
```

3．隐式提交

当遇到 DDL 语句时，DM 数据库会自动提交前面的事务，然后开始一个新的事务执行 DDL 语句。这种事务提交被称为隐式提交。DM 数据库在遇到以下 SQL 语句时自动提交前面的事务：

1) CREATE；
2) ALTER；
3) TRUNCATE；
4) DROP；
5) GRANT；
6) REVOKE；
7) 审计设置语句。

9.1.3 保存点相关语句

SAVEPOINT 语句用于在事务中设置保存点。保存点提供了一种灵活的回滚，事务在执行中可以回滚到某个保存点，在该保存点以前的操作有效，而以后的操作被回滚掉。一个事务中可以设置多个保存点。

1．设置保存点

语法格式

```
SAVEPOINT <保存点名>;
```

参数

<保存点名>　　指明保存点的名字。

使用说明

一个事务中可以设置多个保存点，但不能重名。

2. 回滚到保存点

语法格式

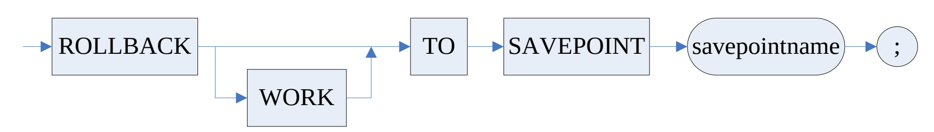
```
ROLLBACK [WORK] TO SAVEPOINT <保存点名>;
```

参数

(1) WORK　　支持与标准 SQL 的兼容性，ROLLBACK 和 ROLLBACK WORK 等价；

(2) <保存点名> 指明部分回滚时要回滚到的保存点的名字。

图例



使用说明

回滚到保存点后事务状态和设置保存点时事务的状态一致，在保存点以后对数据库的操作被回滚。

举例说明

例 插入数据到表 ADDRESS_TYPE 后设置保存点，然后再插入另一数据，回滚到保存点。

```
(1)往表 ADDRESS_TYPE 中插入一个数据
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('发货地址');
(2)查询表 ADDRESS_TYPE
SELECT * FROM PERSON.ADDRESS_TYPE;
/*地址类型名为'发货地址'的记录已经被插入到表中*/
(3)设置保存点
SAVEPOINT A;
(4)往表 ADDRESS_TYPE 中插入另一个数据
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('家庭地址');
(5)回滚到保存点
ROLLBACK TO SAVEPOINT A;
(6)查询表 ADDRESS_TYPE
SELECT * FROM PERSON.ADDRESS_TYPE;
/*插入操作被回滚，ADDRESS_TYPE 中不存在地址类型名为'家庭地址'的记录*/
```

9.1.4 设置事务隔离级及读写特性

事务的隔离级描述了给定事务的行为对其它并发执行事务的暴露程度。通过选择三个隔离级中的一个，用户能增加对其它未提交事务的暴露程度，获得更高的并发度。DM 允许用户改变未启动的事务的隔离级和读写特性，即下列语句必须在事务开始时执行，否则无效。

1. 设置事务隔离级语句

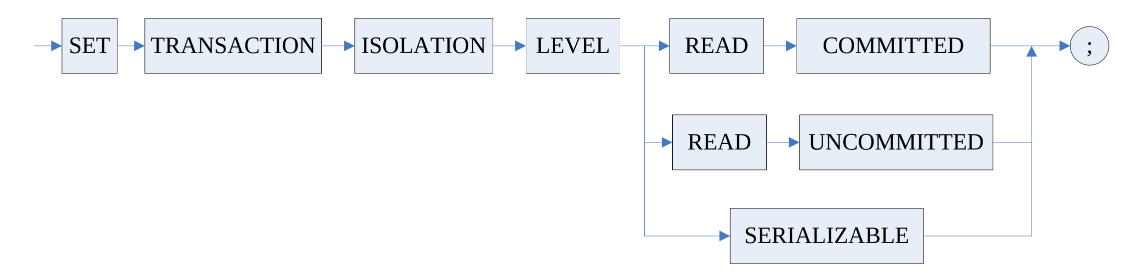
事务的隔离级描述了给定事务的行为对其它并发执行事务的暴露程度。通过选择三个隔

离级中的一个，用户能增加对其它未提交事务的暴露程度，获得更高的并发度。

语法格式

```
SET TRANSACTION ISOLATION LEVEL <事务隔离级>;
<事务隔离级> ::= READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE
```

图例



使用说明

1） 该语句设置事务的隔离级别：

----读提交(READ COMMITTED)：DM 默认级别，保证不读脏数据；

----读未提交(READ UNCOMMITTED)：可能读到脏数据；

----可串行化(SERIALIZABLE)：事务隔离的最高级别，事务之间完全隔离。

一般情况下，使用读提交隔离级别可以满足大多数应用，如果应用要求可重复读以保证基于查询结果的更新的正确性就必须使用可重复读或可串行读隔离级别。在访问只读表和视图的事务，以及某些执行 SELECT 语句的事务(只要其他事务的未提交数据对这些语句没有负面效果)时，可以使用读未提交隔离级。

2） 只能在事务未开始执行前设置隔离级，事务执行期间不能更改隔离级。

2. 设置事务读属性的语句

语法格式

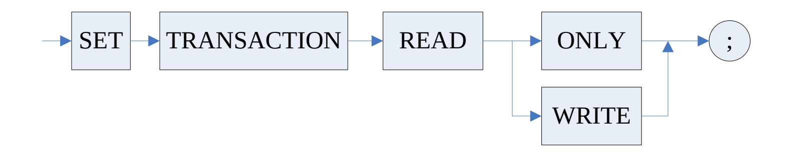
```
SET TRANSACTION <事务读写属性>;
<事务读写属性> ::= READ ONLY | READ WRITE
```

参数

(1)READ ONLY　　只读事务，该事务只能做查询操作，不能更新数据库；

(2)READ WRITE　　读写事务，该事务可以查询并更新数据库，是 DM 的默认设置。

图例



语句功能

该语句设置事务的读写属性。

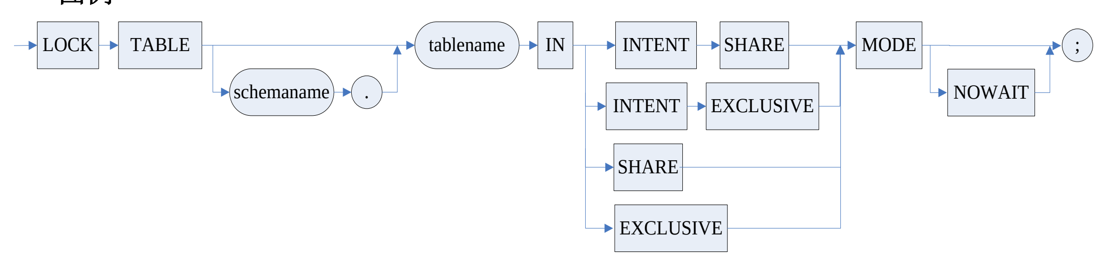
9.2 DM 手动上锁语句

DM 的隐式封锁足以保证数据的一致性，但用户可以根据自己的需要手工显式锁定表，允许或禁止在当前用户操作期间其它用户对此表的存取。DM 提供给用户四种表锁的封锁：意向共享锁(IS)、共享锁(S)、意向排他锁(IX)和排他锁(X)。

语法格式

```
LOCK TABLE [<模式名>.]<表名> IN <封锁方式> MODE [NOWAIT];
<封锁方式>::=
[INTENT SHARE ]|
[INTENT EXCLUSIVE] |
SHARE |
EXCLUSIVE
```

图例



使用说明

1． 意向共享表封锁：INTENT SHARE TABLE LOCKS (IS)

该封锁表明该事务封锁了表上的一些元组并试图修改它们(但是还未做修改，其它事务可读这些元组，但是不能修改这些元组)。意向共享表封锁是限制最少的锁，提供了表上最大的并发度。

1) 允许操作：

其他事务对该表的并发查询、插入、更新、删除或在该表上进行封锁，其他事务可以同时上意向共享锁(IS)、意向排他锁(IX)、共享锁(S)。

2) 禁止操作：

其它事务以排他锁方式(X)存取该表。

```
LOCK TABLE tablename IN EXCLUSIVE MODE;
```

2． 意向排他表封锁：INTENT EXCLUSIVE TABLE LOCKS (IX)

该锁表明该事务对表的元组进行一次或多次修改(其它事务不能访问这些元组)， 行排他表封锁较行共享表封锁稍严格。

1) 允许操作：

其它事务并行查询、插入、更新、删除或封锁该表上行，允许多个事务在同一表上获得意向排他(IX)和意向共享(IS)表锁。

2) 禁止操作：

其它事务对表进行共享(S)或者排他读写封锁(X)。

```
LOCK TABLE tablename IN SHARE MODE;
LOCK TABLE tablename IN EXCLUSIVE MODE;
```

3． 共享表封锁：SHARE TABLE LOCKS (S)

该锁表明该事务访问表中所有元组，其他事务不能对该表做任何更新操作。

1) 允许操作：

其它事务在该表上作查询，但是不允许作修改，且允许多个事务在同一表上并发地持有共享表封锁(S)。

2) 禁止操作：

其它事务对表进行执行意向排他锁(IX)和排他锁方式(X)封锁。

```
LOCK TABLE tablename IN INTENT EXCLUSIVE MODE;
LOCK TABLE tablename IN EXCLUSIVE MODE;
```

4． 排他表封锁：EXCLUSIVE TABLE LOCKS (X)

该封锁是表封锁中最严格的方式，只允许持有封锁的事务可对该表进行修改。

1） 允许操作：

不允许任何操作。

2） 禁止操作：

其它事务对表执行任何 DML 语句，即不能插入、修改和删除该表中的行，封锁该表中的行或以任何方式封锁表。

用户上锁成功后锁将一直有效，直到当前事务结束时，该锁被系统自动解除；

5． 当使用 NOWAIT 时，若不能立即上锁成功则立刻返回报错信息，不再等待。

举例说明

例 当用户 SYSDBA 希望独占某表，他可以对该表显式地上排他锁。

```
LOCK TABLE PERSON.ADDRESS IN EXCLUSIVE MODE;
```

第 10 章 外部函数

为了能够在创建和使用自定义 DMSQL 程序时，使用其他语言实现的接口，DM8 提供了C、JAVA 外部函数，这样即使外部函数在执行中出现了任何问题，都不会影响到服务器的正常执行。

需要注意的是，无论是 C 外部函数还是 JAVA 外部函数，都需要将动态库或 jar 包上传到服务器端或在服务器端编译生成动态库或 jar 包，系统管理员应对动态库和 jar 包进行严格审查，以防止外部函数中包含病毒或恶意代码，引发安全问题。为了保证数据库的安全性和灵活性，DM 提供了 ini 参数 ENABLE_EXTERNAL_CALL 来开关外部函数功能，默认情况下，数据库会关闭外部函数的创建和执行功能。

10.1 C 外部函数

C 外部函数是使用 C、C++语言编写，在数据库外编译并保存在.dll、.so 共享库文件中，被用户通过 DMSQL 程序调用的函数。

C 外部函数的执行都通过代理 dmap 工具进行，为了执行 C 外部函数，需要先启动 dmap 服务。dmap 执行程序在 DM8 安装目录的 bin 子目录下，直接执行即可启动 dmap 服务。

当用户调用 C 外部函数时，服务器操作步骤如下：首先，确定调用的（外部函数使用的）共享库及函数；然后，通知代理进程工作。代理进程装载指定的共享库，并在函数执行后将结果返回给服务器。

10.1.1 生成动态库

DM8 提供两种方案编写 C 外部函数：

■ **DM 结构化参数**

该方案中，用户必须使用 DM8 提供的编写 C 外部函数动态库的接口，严格按照如下格式书写外部函数的 C 代码。

C 外部函数格式

```
de_data 函数名(de_args *args)
{
    C 语言函数实现体;
}
```

参数

1. <de_data> 返回值类型。de_data 结构体类型如下：

```
struct de_data{
    int  null_flag;   /*参数是否为空，1 表示非空，0 表示空*/
    union          /*只能为 int、double 或 char 类型*/
   {
        int    v_int;
        double  v_double;
        char   v_str[];
    }data;
};
```

2. <de_args> 参数信类型。de_args 结构体类型如下：

```
struct de_args
```

```
{
    int         n_args;       /*参数个数*/
   de_data*      args;         /*参数列表*/
};
```

3. < C 语言函数实现体> C 语言函数对应的函数实现体。

使用说明

1. C 语言函数的参数可通过调用 DM8 提供的一系列 get 函数得到，同时可调用 set 函数重新设置这些参数的值；
2. 根据返回值类型，调用不同的 return 函数接口；
3. 必须根据参数类型、返回值类型，调用相同类型的 get、set 和 return 函数。当调用 de_get_str 和 de_get_str_with_len 得到字符串后，必须调用 de_str_free 释放空间；
4. DM8 提供的编写 C 外部函数动态库的接口如表 10.1 所示。

表 10.1 DM8 支持的编写 C 外部函数动态库的接口

函数类型	函数名	功能说明
get	int de_get_int(de_args *args, int arg_id);	第arg_id参数的数据类型为整型，从参数列表 args 中取出第 arg_id 参数的值，
	double de_get_double(de_args *args, int arg_id);	第arg_id参数的数据类型为 double 类型，从参数列表 args 中取出第 arg_id 参数的值。
	char* de_get_str(de_args *args, int arg_id);	第arg_id参数的数据类型为字符串类型，从参数列表 args 中取出第 arg_id 参数的值。
	char* de_get_str_with_len(de_args *args, int arg_id, int* len);	第arg_id参数的数据类型为字符串类型，从参数列表 args 中取出第 arg_id 参数的值以及字符串长度。
set	void de_set_int(de_args *args, int arg_id, int ret);	第arg_id参数的数据类型为整型，设置参数列表 args 的第 arg_id 参数的值为 ret。
	void de_set_double(de_args *args, int arg_id, double ret);	第arg_id参数的数据类型为 double 类型，设置参数列表 args 的第 arg_id 参数的值为 ret。
	void de_set_str(de_args *args, int arg_id, char* ret);	第arg_id参数的数据类型为字符串类型，设置第 arg_id 参数的值为 ret。
	void de_set_str_with_len(de_args *args, int arg_id, char* ret, int	第arg_id参数的数据类型为字符串类型，将字符串

	len);	ret 的前 len 个字符赋值给参数列表 args 的第 arg_id 参数。
	void de_set_null(de_args *args, int arg_id);	设置参数列表 args 的第 arg_id 个参数为空。
return	de_data de_return_int(int ret);	返回值类型为整型。
	de_data de_return_double(double ret);	返回值类型为 double 型。
	de_data de_return_str(char* ret);	返回值为字符串类型。
	de_data de_return_str_with_len(char* ret, int len);	返回字符串 ret 的前 len 个字符。
	de_data de_return_null();	返回空值。
de_str_free	void de_str_free(char* str);	调用 de_get_str 函数后，需要调用此函数释放字符串空间。
de_is_null	int de_is_null(de_args *args, int arg_id);	判断参数列表 args 的第 arg_id 个参数是否为空。

注：参数个数 arg_id 的起始值为 0。

■ **标量类型参数**

该方案中，用户不必引用 DM 提供的外部函数接口，可以按照标准的 C 风格编码，使用 C 标量类型作为参数类型。使用该方案编写的 C 函数，只能在使用 X86 CPU 的 64 位非 Windows 系统中，被数据库引用作为外部函数。

C 外部函数格式

```
返回类型函数名(参数列表)
{
    C 语言函数实现体;
}
```

使用说明

1. 返回类型及参数列表中参数的数据类型只支持 int、double 以及 char*类型；
2. 参数列表中不支持 out 型参数；
3. 如果参数列表中有 char*的参数，不必在函数中对其进行释放；为了安全考虑，最好只对其进行只读操作；
4. 如果返回 char*类型，返回值必须使用 malloc 申请空间，且必须有结尾 0，不允许直接返回常量或返回参数列表中传入的字符类型参数。

10.1.2 C 外部函数创建

语法格式

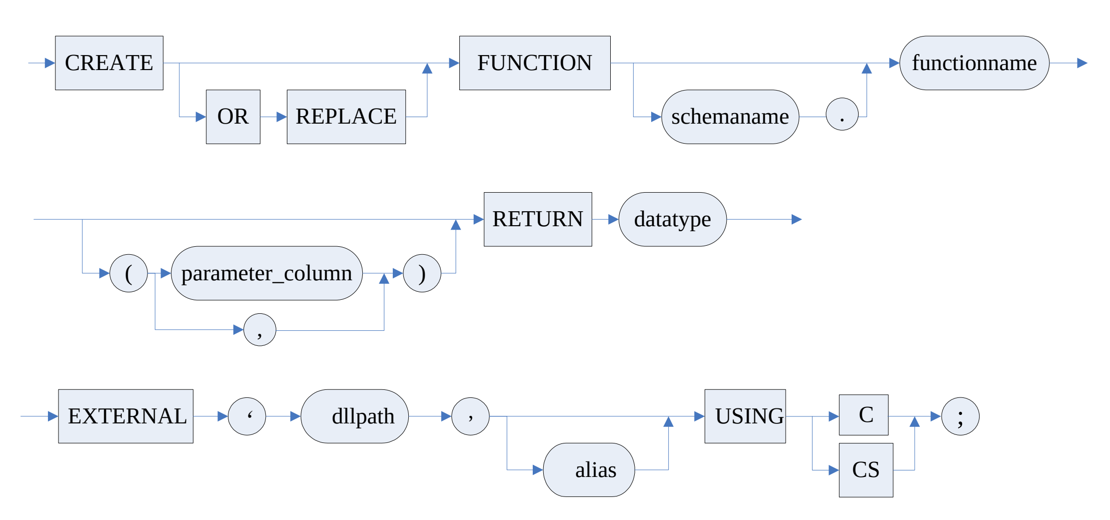
```
CREATE OR REPLACE FUNCTION [<模式名>.]<函数名>[(<参数列表>)]
RETURN <返回值类型>
EXTERNAL '<动态库路径>' [<引用的函数名>] USING < C | CS >;
```

参数

1. <函数名>　　指明被创建的 C 外部函数的名字；

2. <模式名> 指明被创建的 C 外部函数所属模式的名字，缺省为当前模式名；
3. <参数列表> 指明 C 外部函数参数信息，如果是使用 DM 结构化参数编写的 C 函数，参数模式可设置为 IN、OUT 或 IN OUT（OUT IN），缺省为 IN 类型；而使用标量类型参数编写的 C 函数则参数模式只能是 IN 类型。参数类型、个数都应和动态库里定义的一致；
4. <返回值类型> 必须和动态库里定义的一致；
5. <动态库路径> 用户按照 DM 规定的 C 语言函数格式编写的 DLL 文件生成的动态库所在的路径；动态库分为 64 位和 32 位两种，使用的时候要和操作系统一一对应。例如，64 位的操作系统要用 64 位的动态库；
6. <引用函数名> 指明<函数名>在<动态库路径>中对应的函数名；
7. USING 子句指明函数的类型，如果是 DM 结构化参数的 C 函数，类型为 C；标量类型参数的 C 函数，类型为 CS。

图例



语句功能

创建自定义 C 外部函数。

使用说明

1. 仅允许 DBA 创建 C 外部函数；
2. <引用函数名>如果为空，则默认与<函数名>相同；
3. <动态库路径>分为.dll 文件（windows）和.so 文件（linux）两种；
4. 外部函数定义的返回值类型与对应 C 语言函数的返回值类型必须一致，否则会导致返回值不可预测甚至系统异常。

权限

使用该语句的用户必须是 DBA 或该存储过程的拥有者且具有 CREATE FUNCTION 数据库权限的用户。

10.1.3 举例说明

例 编写（C 语言）外部函数 C_CONCAT，用于将两个字符串连接。

可以使用 DM 结构化参数方案来完成。

生成动态库。

第一步，使用 Microsoft Visual Studio 2008 创建新项目 newp，位于 d:\xx\tt

文件夹中。在 d:\xx\tt\newp\newp 文件夹中,直接拷入 dmde.lib 动态库和 de_pub.h 头文件。dmde.lib 和 de_pub.h 位于安装文件 x:\...\dmdbms\include 中。

第二步，在 newp 项目中，添加头文件。将已有 de_pub.h 头文件添加进来，同时，添加新的 tt.h 头文件。tt.h 文件内容如下：

```
#include "de_pub.h"
#include "string.h"
#include "stdlib.h"
```

第三步，在 newp 项目中，添加源文件。名为 tt.c。tt.c 内容如下：

```
#include "tt.h"
de_data C_CONCAT(de_args *args)
{
    de_data  de_ret;
    char* str1;
    char* str2;
    char* str3;
    int len1;
    int len2;

    str1 = (char*)de_get_str(args, 0);                    /*从参数列表中取第 0 个参数*/
    str2 = (char*)de_get_str_with_len(args, 1, (udint4*)&len2);        /*从参数列表中取第 1 个参数的值以及长度*/
    len1 = strlen(str1);
    str3 = (char*)malloc(len1 + len2);
    memcpy(str3, str1, len1);
    memcpy(str3 + len1, str2, len2);

    de_str_free((sdbyte*)str1);     /*调用 get 函数得到字符串之后，需要调用此函数释放字符串空间*/
    de_str_free((sdbyte*)str2);

    de_ret = de_return_str_with_len((udbyte*)str3, len1 + len2);    /*返回字符串*/
    free(str3);
    return de_ret;
}
```

第四步，在 newp 项目的源文件中，添加模块定义文件 tt.def，内容如下：

```
LIBRARY "tt.dll"
EXPORTS
    C_CONCAT
```

第五步，在 Microsoft Visual Studio 2008 界面上，单击项目，找到 newp 属性，点击打开。在"配置属性–链接器–输入"中添加附加依赖项 dmde.lib，在"配置属性–常规"中调整配置类型为动态库（.dll）。

第六步，编译 tt 项目（设置项目输出文件路径：D:\xx\tt）。得到 newp.dll 文件。

至此，外部函数的使用环境准备完毕。

其次，创建并使用外部函数。

第一步，启动数据库服务器 dmserver，启动 disql。（dmserver、disql 等工具位于安装目录 x:\...\dmdbms\bin 中）

第二步，在 disql 中，创建外部函数 MY_CONCAT，语句如下：

```
CREATE OR REPLACE FUNCTION MY_CONCAT(A VARCHAR, B VARCHAR)
RETURN VARCHAR
EXTERNAL 'd:\xx\tt\newp.dll' C_CONCAT USING C;
```

第三步，调用 C 外部函数，语句如下：

```
select MY_CONCAT ('hello ', 'world!');
```

第四步，查看结果：

```
hello world!
```

也可以使用 C 标量类型参数方案完成，不过需要注意该方案仅支持使用 X86 CPU 的 64 位非 Windows 系统。

生成动态库，这里改用 LINUX 操作系统作为示例。

第一步，创建 C 文件 test.c，编写 C 函数

```
#include <string.h>
#include "stdlib.h"
char*  C_CONCAT (char*  str1, char* str2)
{
    char* str3;
    int len1;
    int len2;
    len1 = strlen(str1);
    len2 = strlen(str2);
    str3 = (char*)malloc(len1 + len2 + 1);  //要多一个字节作为结尾 0
    memcpy(str3, str1, len1);
    memcpy(str3 + len1, str2, len2);
    str3[len1 + len2] = 0;  //必须有结尾 0
    return str3;
}
```

第二步，生成动态库

gcc -o /mnt/libtest.so -fPIC -shared test.c

获得 libtest.so 文件

至此，外部函数的使用环境准备完毕。

其次，创建并使用外部函数。

第一步，启动数据库服务器 dmserver，启动 disql。

第二步，在 disql 中，创建外部函数 MY_CONCAT，语句如下：

```
CREATE OR REPLACE FUNCTION MY_CONCAT(A VARCHAR, B VARCHAR)
RETURN VARCHAR
EXTERNAL '/mnt/libtest.so' C_CONCAT USING CS;
```

第三步，调用 C 外部函数，语句如下：

```
select MY_CONCAT ('hello ', 'world!');
```

第四步，查看结果：

```
hello world!
```

10.2 JAVA 外部函数

JAVA 外部函数是使用 JAVA 语言编写，在数据库外编译生成的 jar 包，被用户通过 DMSQL 程序调用的函数。

JAVA 外部函数的执行都通过代理 dmagent 工具进行，为了执行 JAVA 外部函数，需要先启动 dmagent 服务。dmagent 执行程序在 DM8 安装目录的 tool/dmagent 子目录下，其使用说明文档可参看该目录下的《readme》文档。

当用户调用 JAVA 外部函数时，服务器操作步骤如下：首先，确定调用（外部函数使用的）jar 包及函数；然后，通知代理进程工作。代理进程装载指定的 jar 包，并在函数执行后将结果返回给服务器。

需要注意的是，进行 JAVA 外部函数调用应保证当前用户可以运行 JAVA 命令，否则会导致调用失败。

10.2.1 生成 jar 包

用户必须严格按照 JAVA 语言的格式书写代码，完成后生成 jar 包。

10.2.2 JAVA 外部函数创建

语法格式

```
CREATE OR REPLACE FUNCTION [<模式名>.]<函数名>[(<参数列表>)]
RETURN <返回值类型>
EXTERNAL '<jar 包路径>' [<引用的函数名>] USING JAVA;
```

参数

1． <函数名>　　被创建的 JAVA 外部函数的名字；

2． <模式名>　　被创建的 JAVA 外部函数所属模式的名字，缺省为当前模式名；

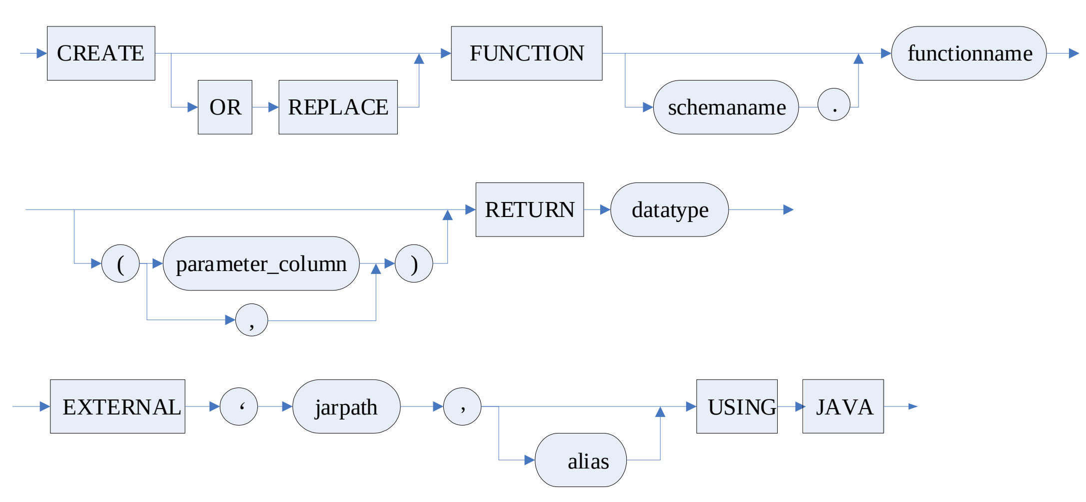
3． <参数列表>　　JAVA 外部函数参数信息，参数模式可设置为 IN、OUT 或 IN OUT（OUT IN），缺省为 IN 类型。参数类型、个数都应和 jar 包里的一致。目前支持的函数参数类型：Int、字符串(char、varchar、varchar2)、bigint、double。分别对应 java 类型：Int、string、 long、double；

4． <返回值类型>　必须 jar 包里定义的一致；

5． <jar 包路径>　用户按照 java 语言格式编写的源码生成的 jar 包，及其所依赖的 jar 包所在的绝对路径。dmagent 方式 jar 包要使用绝对路径；ap 方式 jar 包名字是不起作用的；

6． <引用函数名>　指明<函数名>在<jar 包路径>中对应的函数名，jar 包中的函数应为 static 类型。若<函数名>包含包名，则包名（或.包名）与类名的分隔符要用.或/。而类与方法的分隔符要用.。

图例



语句功能

创建自定义 JAVA 外部函数。

使用说明

1． 仅允许 DBA 创建 JAVA 外部函数；

2． <引用函数名>书写格式：包名、类名与方法名之间用.进行分隔。

权限

使用该语句的用户必须是 DBA 或该存储过程的拥有者且具有 CREATE FUNCTION 数据

库权限的用户。

10.2.3 举例说明

例 编写（JAVA 语言）外部函数：testAdd 用于求两个数之和，testStr 用于在一个字符串后面加上 hello。

首先，生成 jar 包。

第一步，使用 eclipse 创建新项目 newp，位于 F:\workspace 文件夹中。

第二步，在 newp 项目中，添加包。右击 newp，新建（new）一个 package，命名（name）为 com.test.package1。

第三步，在 package1 包中，添加类文件。右击 src，新建（new）一个 class，命名（name）为 test。Modifiers 选择 public。class 文件内容如下：

```
package com.test.package1;
public class test {
    public static int testAdd(int a, int b) {
        return a + b;
    }
    public static String testStr(String str) {
        return str + " hello";
    }
 }
```

第四步，生成 jar 包。在 newp 项目中，右击，选中 EXPORT，选择 Java\JAR file，取消.classpath 和.project 的勾选。目标路径 JAR file 设置为：E:\test.jar，然后 finish。

第五步，查看 E 盘中 test.jar。已经存在。

第六步，在安装目录的..\dmdbms\bin 中创建一个 external_jar 文件夹，将 test.jar 拷入其中。

至此，外部函数的使用环境准备完毕。

其次，创建并使用外部函数。

第一步，启动数据库服务器 dmserver，启动 DM 管理工具。

第二步，在 DM 管理工具中，创建外部函数 MY_INT 和 MY_chr，语句如下：

```
CREATE OR REPLACE FUNCTION MY_INT(a int, b int)
RETURN int
EXTERNAL                              '..\dmdbms\bin\external_jar\test.jar'
"com.test.package1.test.testAdd" USING java;

CREATE OR REPLACE FUNCTION MY_chr(s varchar)
RETURN varchar
EXTERNAL                              '..\dmdbms\bin\external_jar\test.jar'
"com.test.package1.test.testStr" USING java;
```

第三步，调用 JAVA 外部函数，语句如下：

```
select MY_INT(1,2);
select MY_chr('abc');
```

第四步，查看结果，分别为：

```
3
abc hello
```

10.3 DMAP 使用说明

DMAP（DM Assit progress）作为数据库管理系统的辅助进程，提供外部函数、备份还原等功能的执行。

10.3.1 启动 DMAP

安装 DM 数据库以后，DMAP 服务会自动启动。如果需要手动启动，有两种途径：一是启动 DM 服务查看器中的 DmAPService；二是通过手动启动 DMAP 执行码实现，DMAP 执行码位于 DM 安装目录的 bin 子目录下。除此之外，LINUX 下，还可以调用 bin 目录下的 DmAPService 脚本启动 DMAP 服务。

默认不带参数启动 DMAP 时，DMAP 监听端口号为 4236，与 DM 服务器 INI 参数 EXTERNAL_AP_PORT 的默认值一致。

也可以使用参数 dmap_ini 指定 dmap.ini 配置文件启动 DMAP，如下例所示：

```
dmap  dmap_ini=d:\dmap.ini
```

其中，dmap.ini 可配置 dmap 的端口号，如下所示：

```
AP_PORT=4236
```

需要注意的是，当使用 dmap.ini 对 AP_PORT 进行配置时，DM 服务器 INI 参数 EXTERNAL_AP_PORT 的配置值应与 AP_PORT 一致。

10.3.2 使用 DMAP 执行外部函数

例：

```
CREATE OR REPLACE FUNCTION MY_CONCAT(A VARCHAR, B VARCHAR)
RETURN VARCHAR
EXTERNAL 'D:\testroot\for_dmserver\smoketest_data\dameng\detest64.dll' C_CAT
USING C;
/
SQL> select MY_CONCAT ('hello ', 'a');


行号      MY_CONCAT('hello','a')
---------- ----------------------
1         hello a
```

可在 log 目录下查看 dmap.log 日志文件，其中记录了执行外部函数时消息收发的日志。

第 11 章 包

DM 支持 DMSQL 程序包来扩展数据库功能，用户可以通过包来创建应用程序或者使用包来管理过程和函数。

11.1 创建包

包的创建包括包规范和包主体的创建。

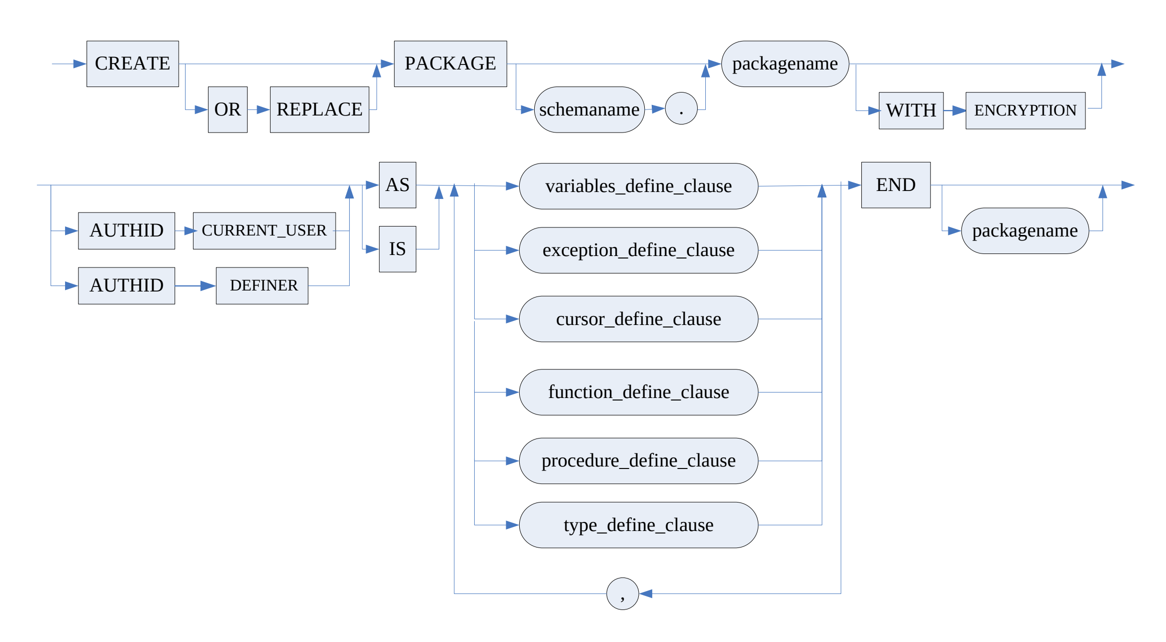
11.1.1 创建包规范

包规范中包含了有关包中的内容信息，但是它不包含任何过程的代码。定义一个包规范的详细语法如下。

语法格式

```
CREATE [OR REPLACE] PACKAGE [<模式名>.]<包名>[WITH ENCRYPTION] [AUTHID DEFINER | AUTHID CURRENT_USER] AS|IS <包内声明列表> END [包名]
<包内声明列表> ::= <包内声明>;{<包内声明>;}
<包内声明> ::= <变量列表定义>|<游标定义>|<异常定义>|<过程定义>|<函数定义>|<类型声名>
<变量列表定义>::= <变量定义>{<变量定义>}
<变量定义> ::= <变量名><变量类型>[DEFAULT|ASSIGN|:=<表达式>]
<变量类型>::=<DMSQL 程序类型>|< [模式名.]表名.列名%TYPE>|<[模式名.]表名%ROWTYPE>|<记录类型>
<记录类型>::= RECORD(<变量名> <DMSQL 程序类型>;{<变量名> <DMSQL 程序类型>;})
<游标定义> ::= CURSOR <游标名> [FOR <查询语句>]
<异常定义> ::= <异常名> EXCEPTION [FOR <异常码>]
<过程定义> ::= PROCEDURE <过程名> <参数列表>
<函数定义> ::= FUNCTION <函数名><参数列表> RETURN <返回值数据类型>[RESULT_CACHE] [DETERMINISTIC] [PIPELINED]
<类型声名> ::= TYPE <类型名称> IS <数据类型>
```

图例



使用说明

1. 包部件可以以任意顺序出现，其中的对象必须在引用之前被声明；
2. 过程和函数的声明都是前向声明，包规范中不包括任何实现代码。

权限

1、使用该语句的用户必须是 DBA 或该包对象的拥有者且具有 CREATE PACKAGE 数据库权限的用户；

2、可以用关键字 AUTHID DEFINER |AUTHID CURRENT_USER 指定包的调用者权限，若为 DEFINER，则采用包定义者权限，若为 CURRENT_USER 则为当前用户权限，默认为包定义者权限。

11.1.2 创建包主体

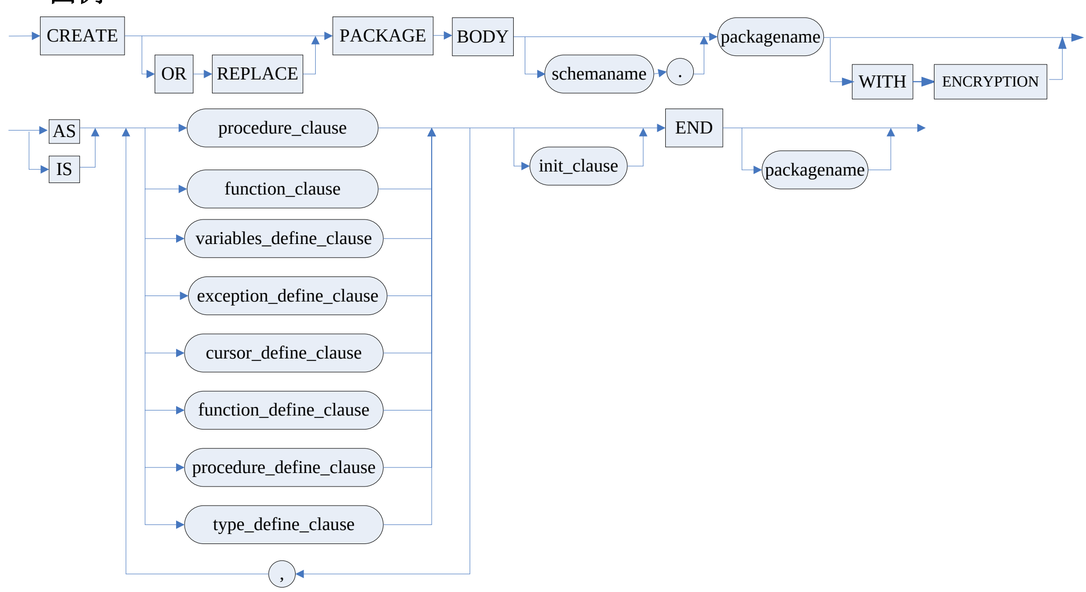
包主体中包含了在包规范中的前向子程序声明相应的代码。它的创建语法如下。

语法格式

```
CREATE [OR REPLACE] PACKAGE BODY [<模式名>.]<包名> [WITH ENCRYPTION] AS|IS <包体部分> END [包名]
<包体部分> ::= <包体声明列表> [<初始化代码>]
<包体声明列表> ::=<包体声明>[,{<包体声明> }……]
<包体声明>::=<变量定义>|<游标定义>|<异常定义>|<过程定义>|<函数定义>|<类型声名> |<存储过程实现>|<函数实现>
<变量定义> ::= <变量名列表> <数据类型> [默认值定义]
<游标定义> ::= CURSOR <游标名> [FOR <查询语句>]
<异常定义> ::= <异常名> EXCEPTION [FOR <异常码>]
<过程定义> ::= PROCEDURE <过程名> <参数列表>
<函数定义> ::= FUNCTION <函数名> <参数列表> RETURN <返回值数据类型>
<类型声明> ::= TYPE <类型名称> IS <数据类型>
<存储过程实现> ::= PROCEDURE <过程名> <参数列表> AS|IS BEGIN <实现体> END [过程名];
<函数实现> ::= FUNCTION <函数名><参数列表> RETURN <返回值数据类型>[DETERMINISTIC]
```

```
[PIPELINED]<AS|IS> BEGIN <实现体> END [函数名];
    <初始化代码> ::= [[<说明部分>]BEGIN<执行部分>[<异常处理部分>]]
    <说明部分> ::=[DECLARE]<说明定义>{<说明定义>}
    <说明定义>::=<变量列表说明>|<异常变量说明>|<游标定义>|<子过程定义>|<子函数定义>;
    <变量列表说明>::= <变量初始化>{<变量初始化>}
    <记录类型>::= RECORD(<变量名> <DMSQL 程序类型>;{<变量名> <DMSQL 程序类型>;})
    <异常变量说明>::=<异常变量名>EXCEPTION[FOR<错误号>]
    <游标定义>::=CURSOR <游标名> [FOR<查询表达式>|<表连接>]
    <子过程定义>::=PROCEDURE<过程名>[(<参数列>)]<IS|AS><模块体>
    <子函数定义>::=FUNCTION<函数名>[(<参数列>)]RETURN<返回数据类型><IS|AS><模块体>
    <执行部分>::=<SQL 过程语句序列>{< SQL 过程语句序列>}
    < SQL 过程语句序列>::=[<标号说明>]<SQL 过程语句>;
    <标号说明>::=<<<标号名>>>
    <SQL 过程语句>::=<SQL 语句>|<SQL 控制语句>
    <异常处理部分>::=EXCEPTION<异常处理语句>{<异常处理语句>}
    <异常处理语句>::= WHEN <异常名> THEN < SQL 过程语句序列>;
```





使用说明

1．包规范中定义的对象对于包主体而言都是可见的，不需要声明就可以直接引用。这些对象包括变量、游标、异常定义和类型定义；

2．包主体中不能使用未在包规范中声明的对象；

3．包主体中的过程、函数定义必须和包规范中的前向声明完全相同。包括过程的名字、参数定义列表的参数名和数据类型定义；

4．包中可以有重名的过程和函数，只要它们的参数定义列表不相同。系统会根据用户的调用情况进行重载(OVERLOAD)；

5．用户在第一次访问包（如调用包内过程、函数，访问包内变量）时，系统会自动将包对象实例化。每个会话根据数据字典内的信息在本地复制包内变量的副本。如果用户定义了 PACKAGE 的初始化代码，还必须执行这些代码(类似于一个没有参数的构造函数执行)；

6．对于一个会话，包头中声明的对象都是可见的，只要指定包名，用户就可以访问这些对象。可以将包头内的变量理解为一个 SESSION 内的全局变量；

7．关于包内过程、函数的调用：DM 支持按位置调用和按名调用参数两种模式。除了需要在过程、函数名前加入包名作为前缀，调用包内的过程、函数的方法和普通的过程、函数并无区别；

8．包体内声明的变量、类型、方法以及实现的未在包头内声明的方法被称作本地变量、方法，本地变量、方法只能在包体内使用，用户无法直接使用；

9．在包体声明列表中，本地变量必须在所有的方法实现之前进行声明；本地方法必须在使用之前进行声明或实现；

10．如果创建包时，在函数定义中使用 DETERMINISTIC 指定该函数为确定性函数，在函数实现中可以省略指定该函数的确定性；如果函数定义中没有指定该函数为确定性函数，则函数实现时不能指定该函数为确定性函数。

11．结果集缓存 RESULT_CACHE，只是语法支持，没有实际意义。

权限

使用该语句的用户必须是 DBA 或该包对象的拥有者且具有 CREATE PACKAGE 数据库权限的用户。

11.2 重编译包

重新对包进行编译，如果重新编译失败，则将包置为禁止状态。

重编功能主要用于检验包的正确性。

语法格式

```
ALTER PACKAGE [<模式名>.]<包名> COMPILE [DEBUG];
```

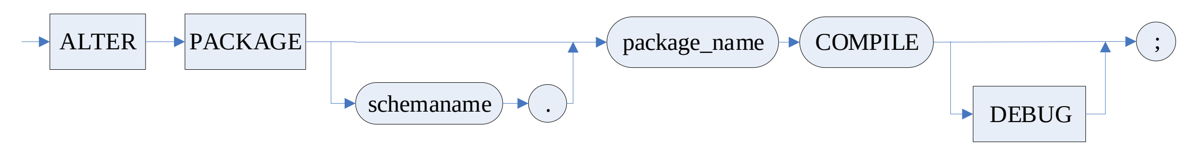
参数

1．<模式名> 指明被重编译的包所属的模式；

2．<包名> 指明被重编译的包的名字；

3．[DEBUG] 可忽略。

图例



权限

执行该操作的用户必须是包的创建者，或者具有 DBA 权限。

11.3 删除包

和创建方式类似，包对象的删除分为包规范的删除和包主体的删除。

11.3.1 删除包规范

从数据库中删除一个包对象。

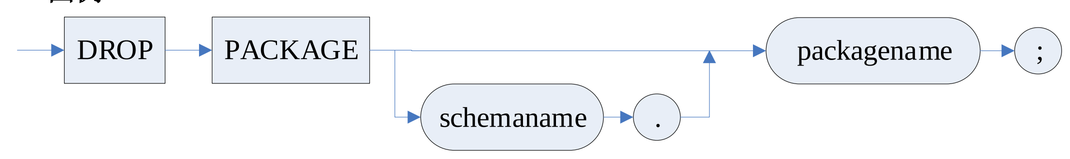
语法格式

```
DROP PACKAGE [<模式名>.]<包名>;
```

参数

1. <模式名>　指明被删除的包所属的模式，缺省为当前模式；
2. <包名>　指明被删除的包的名字。

图例



使用说明

1. 如果被删除的包不属于当前模式，必须在语句中指明模式名；
2. 如果一个包规范被删除，那么对应的包主体被自动删除。

权限

执行该操作的用户必须是该包的拥有者，或者具有 DBA 权限。

11.3.2 删除包主体

从数据库中删除一个包的主体对象。

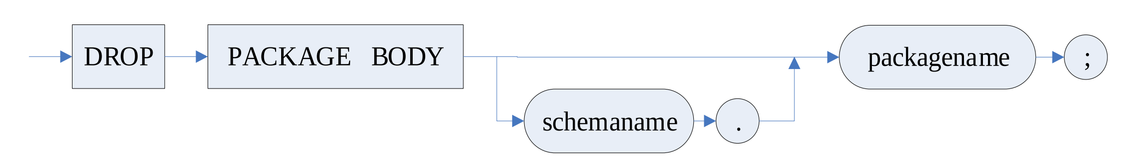
语法格式

```
DROP PACKAGE BODY [<模式名>.]<包名>;
```

参数

1. <模式名>　指明被删除的包所属的模式，缺省为当前模式；
2. <包名>　指明被删除的包的名字。

图例



使用说明

如果被删除的包不属于当前模式，必须在语句中指明模式名。

权限

执行该操作的用户必须是该包的拥有者，或者具有 DBA 权限。

11.4 应用实例

以下是一个包规范的使用实例：

创建实例数据库，这些数据将在之后的例子中用到。

```
CREATE TABLE Person(Id INT IDENTITY, Name VARCHAR(100), City VARCHAR(100));
INSERT INTO Person(Name, City) VALUES('Tom','武汉');
INSERT INTO Person(Name, City) VALUES('Jack','北京');
INSERT INTO Person(Name, City) VALUES('Mary','上海');
```

表中数据如表 11.3.1 所示。

表 11.3.1

ID	NAME	CITY

1	TOM	武汉
2	JACK	北京
3	MARY	上海

创建包规范：

```
  CREATE OR REPLACE PACKAGE PersonPackage AS
       E_NoPerson EXCEPTION;
       PersonCount INT;
       Pcur CURSOR;
       PROCEDURE AddPerson(Pname VARCHAR(100), Pcity varchar(100));
       PROCEDURE RemovePerson(Pname VARCHAR(100), Pcity varchar(100));
       PROCEDURE RemovePerson(Pid INT);
       FUNCTION GetPersonCount RETURN INT;
       PROCEDURE PersonList;
END PersonPackage;
```

这个包规范的部件中包括 1 个变量定义，1 个异常定义，1 个游标定义，4 个过程定义和 1 个函数定义。

以下是一个包主体的实例，它对应于前面的包规范定义，包括 4 个子过程和 1 个子函数的代码实现。在包主体的末尾，是这个包对象的初始化代码。当一个会话第一次引用包时，变量 PersonCount 被初始化为 Person 表中的记录数。

创建包主体：

```
CREATE OR REPLACE PACKAGE BODY PersonPackage AS

  PROCEDURE AddPerson(Pname VARCHAR(100), Pcity varchar(100) )AS
    BEGIN
      INSERT INTO Person(Name, City) VALUES(Pname, Pcity);
      PersonCount = PersonCount + SQL%ROWCOUNT;
    END AddPerson;

  PROCEDURE RemovePerson(Pname VARCHAR(100), Pcity varchar(100)) AS
    BEGIN
      DELETE FROM Person WHERE NAME LIKE Pname AND City like Pcity;
      PersonCount = PersonCount - SQL%ROWCOUNT;
    END RemovePerson;

  PROCEDURE RemovePerson(Pid INT) AS
    BEGIN
      DELETE FROM Person WHERE Id = Pid;
      PersonCount = PersonCount - SQL%ROWCOUNT;
    END RemovePerson;

  FUNCTION GetPersonCount RETURN INT AS
  BEGIN
    RETURN PersonCount;
  END GetPersonCount;
```

```
  PROCEDURE PersonList AS
  DECLARE
    V_id INT;
    V_name VARCHAR(100);
    V_city VARCHAR(100);
  BEGIN
    IF PersonCount = 0 THEN
       RAISE E_NoPerson;
    END IF;
    OPEN Pcur FOR SELECT Id, Name, City FROM Person;
    LOOP
      FETCH Pcur INTO V_id,V_name,V_city;
      EXIT WHEN Pcur%NOTFOUND;
      PRINT ('No.' || (cast (V_id as varchar(100))) || '  ' || V_name || '
来自' || V_city );
    END LOOP;
    CLOSE Pcur;
  END PersonList;

BEGIN
    SELECT COUNT(*) INTO PersonCount FROM Person;
END PersonPackage;
```

重新编译包：

```
ALTER PACKAGE PersonPackage COMPILE;
```

调用包中的 AddPerson 过程，往数据表中增加一条记录：

```
CALL  PersonPackage. AddPerson ('BLACK', '南京') ;
```

当前记录变化如表 11.3.2 所示。

表 11.3.2

ID	NAME	CITY
1	TOM	武汉
2	JACK	北京
3	MARY	上海
4	BLACK	南京

调用包中的 RemovePerson 过程，删除第二条记录：

```
CALL  PersonPackage. RemovePerson ('JACK', '北京') ;
或者
    CALL  PersonPackage. RemovePerson (2) ;
```

在此例中，以上两种写法可以得到相同的结果，系统对同名过程根据实际参数进行了重载。如果过程执行结果没有删除任何一条表中的记录，那么会抛出一个包内预定义的异常：E_NoPerson。

此时表中的数据如表 11.3.3 所示。

表 11.3.3

ID	NAME	CITY

1	TOM	武汉
3	MARY	上海
4	BLACK	南京

引用包中的变量。

```
SELECT PersonPackage. PersonCount;
或者
SELECT PersonPackage. GetPersonCount;
```

以上两句语句的作用是等价的。前一句是直接引用了包内变量，后一句是通过调用包内的子函数来得到想要的结果。

调用包中的过程 PersonList 查看表中的所有记录：

```
CALL  PersonPackage. PersonList;
```

可以得到以下输出：

```
NO.1  Tom 来自武汉
No.3  MARY 来自上海
No.4  BLACK 来自南京
```

第 12 章 类类型

DM 通过类类型在 DMSQL 程序中实现面向对象编程的支持。类将结构化的数据及对其进行操作的过程或函数封装在一起。允许用户根据现实世界的对象建模，而不必再将其抽象成关系数据。

DM 的类类型分为普通类类型和 JAVA CLASS 类型。DM 文档中的示例除了特别声明使用的是 JAVA CLASS 类型，要不然使用的都是普通类类型。

12.1 普通 CLASS 类型

DM 的类的定义分为类头和类体两部分，类头完成类的声明；类体完成类的实现。

类中可以包括以下内容：

1．类型定义

在类中可以定义游标、异常、记录类型、数组类型、以及内存索引表等数据类型，在类的声明及实现中可以使用这些数据类型；类的声明中不能声明游标和异常，但是实现中可以定义和使用。

2．属性

类中的成员变量，数据类型可以是标准的数据类型，可以是在类中自定义的特殊数据类型。

3．成员方法

类中的函数或过程，在类头中进行声明；其实现在类体中完成；

成员方法及后文的构造函数包含一个隐含参数，即自身对象，在方法实现中可以通过 this 或 self 来访问自身对象，self 等价于 this。如果不存在重名问题，也可以直接使用对象的属性和方法。this 和 self 只能在包或对象脚本中调用。

4．构造函数

构造函数是类内定义及实现的一种特殊的函数，这类函数用于实例化类的对象，构造函数满足以下条件：

1） 函数名和类名相同；

2） 函数返回值类型为自身类。

构造函数存在以下的约束：

1） 系统为每个类提供两个默认的构造函数，分别为 0 参的构造函数和全参的构造函数；

2） 0 参构造函数的参数个数为 0，实例的对象内所有的属性初始化值为 NULL；

3） 全参构造函数的参数个数及类型和类内属性的个数及属性相同，按照属性的顺序依次读取参数的值并给属性赋值；

4） 用户可以自定义构造函数，一个类可以有多个构造函数，但每个构造函数的参数个数必须不同；

5） 如果用户自定义了 0 个参数、或参数个数同属性个数相同的构造函数，则会覆盖相应的默认构造函数。

下面从类的声明、类的实现、类的删除、类体的删除和类的使用几部分来详细介绍类类型的实现过程。

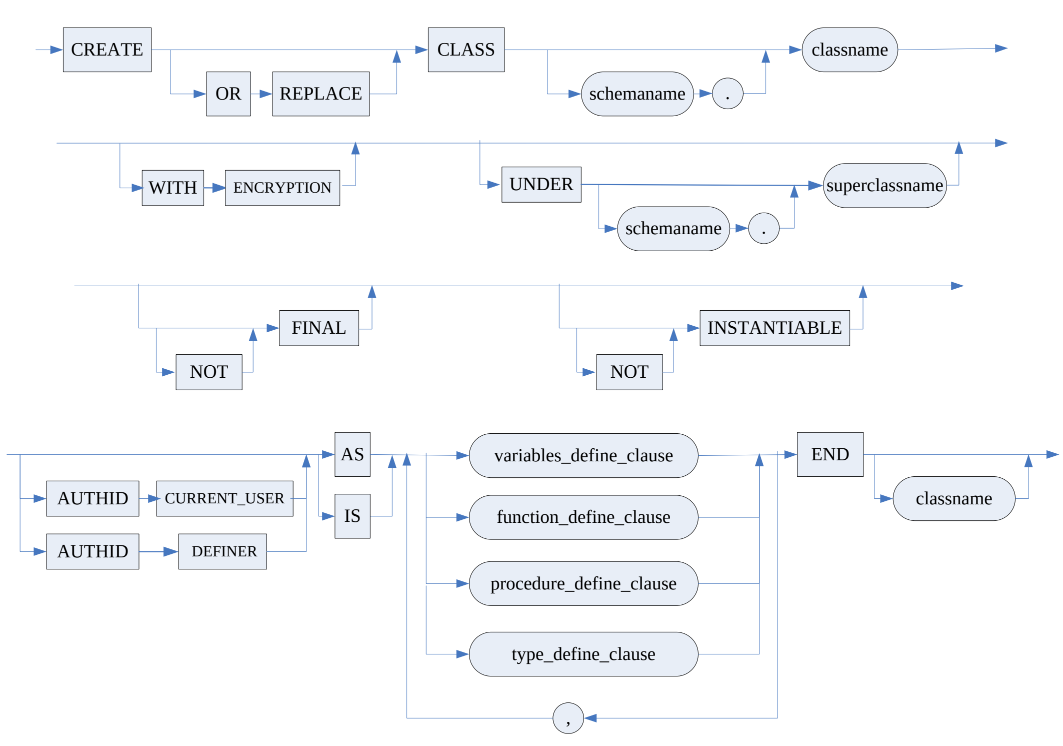
12.1.1 声明类

类的声明在类头中完成。类头定义通过 CREATE CLASS 语句来完成，其语法为：

语法格式

```
CREATE [OR REPLACE] CLASS [<模式名>.]<类名> [WITH ENCRYPTION] [UNDER [<模式名>.]<父类名>] [[NOT] FINAL] [[NOT] INSTANTIABLE] [AUTHID DEFINER | AUTHID CURRENT_USER] AS|IS <类内声明列表> END [类名]
<类内声明列表> ::= <类内声明>;{<类内声明>;}
<类内声明> ::= <变量定义>|<过程定义>|<函数定义>|<类型声名>
<变量定义> ::= <变量名列表> <数据类型> [默认值定义]
<过程定义> ::= [<方法继承属性>][STATIC|MEMBER] PROCEDURE <过程名> <参数列表>
<函数定义> ::= [<方法继承属性>] [MAP] [STATIC|MEMBER] FUNCTION <函数名><参数列表> RETURN <返回值数据类型>[DETERMINISTIC][PIPELINED]
<方法继承属性> ::= <重载属性> | <final 属性> | <重载属性> <final 属性>
<重载属性> ::= [NOT] OVERRDING
<final 属性> ::= FINAL | NOT FINAL | INSTANTIABLE | NOT INSTANTIABLE
<类型声名> ::= TYPE <类型名称> IS <数据类型>
```

图例



使用说明

1. 类中元素可以以任意顺序出现，其中的对象必须在引用之前被声明；
2. 过程和函数的声明都是前向声明，类声明中不包括任何实现代码；
3. 支持对象静态方法声明与调用。可以在 PROCEDURE/FUNCTION 关键字前添加 static 保留字，以此指明方法为静态方法。静态方法只能以对象名为前缀调用，而不能在对象实例中调用；
4. 支持对象成员方法声明与调用。可以在 PROCEDURE/FUNCTION 关键字前添加

MEMBER 以指明方法为成员方法。MEMBER 与 STATIC 不能同时使用，非 STATIC 类型的非构造函数方法默认为成员方法。MAP 表示将对象类型的实例映射为标量数值，只能用于成员类型的 FUNCTION；

5. 关于类继承，有以下使用限制：
    1) 类定义默认为 FINAL，表示该对象类型不能被继承，定义父类时必须指定 NOT FINAL 选项；
    2) 定义子类时必须指定 UNDER 选项；
    3) NOT INSTANTIABLE 对象不能为 FINAL；
    4) NOT INSTANTIABLE 对象不能实例化，但是可以用其子类赋值；
    5) 对象实例化时，必须对父类和子类的成员变量都赋值，且从父类到子类逐个赋值；
    6) 不支持对象的循环继承；
    7) 不支持对象的多继承，即一个类有多个父类；
    8) 不支持父类和子类包含同名变量；
    9) 父类和子类可以同名同参，此时子类必须指定 OVERRIDING；
    10) 方法默认为 NOT OVERRIDING，OVERRIDING 不能与 static 一起使用；
    11) 父类和子类支持同名不同参（参数个数不同、参数个数相同但类型不同）的方法；
    12) 同名且参数个数相同但类型不同时，根据参数类型选择使用的方法；
    13) 方法默认为 INSTANTIABLE，如果声明为 NOT INSTANTIABLE，则不能与 FINAL、STATIC 一起使用；
    14) 如果父类有多个 NOT INSTANTIABLE 方法，子类可以只部分重写，但此时子类必须定义为 NOT FINAL NOT INSTANTIABLE；
    15) NOT INSTANTIABLE 方法不能具有主体；
    16) 方法默认为 NOT FINAL，如果声明为 FINAL，则不能被子类重写；
    17) 子类可以赋值给父类；
    18) 如果父类对应的实例是子类或者子类的孩子，则该父类可以赋值给子类；
    19) 可以用 INSTANTIABLE 子类对 NOT INSTANTIABLE 父类进行赋值；
    20) 子类实例赋值给父类后，调用时使用的是父类方法而不是子类方法；
    21) 支持使用 as 语句转换为父类。

权限

1、使用该语句的用户必须是 DBA 或具有 CREATE CLASS 数据库权限的用户；

2、可以用关键字 AUTHID DEFINER |AUTHID CURRENT_USER 指定类的调用者权限，若为 DEFINER，则采用类定义者权限，若为 CURRENT_USER 则为当前用户权限，默认为类定义者权限。

12.1.2 实现类

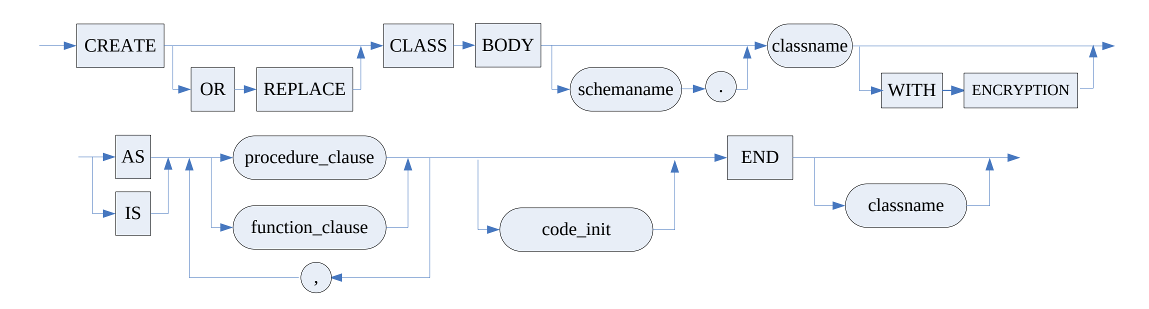
类的实现通过类体完成。类体的定义通过 CREATE CLASS BODY 语句来完成，其语法为：

语法格式

```
CREATE [OR REPLACE] CLASS BODY [<模式名>.]<类名> [WITH ENCRYPTION] AS|IS <类体部分> END [类名]
```

```
<类体部分> ::= <过程/函数列表> [<初始化代码>]
<过程/函数列表> ::= <过程实现|函数实现>{,<过程实现|函数实现> }
<过程实现> ::= [<方法继承属性>][STATIC|MEMBER]PROCEDURE <过程名> <参数列表> AS|IS BEGIN <实现体> END [过程名]
<函数实现> ::= [<方法继承属性>][MAP] [STATIC|MEMBER]FUNCTION <函数名><参数列表> RETURN <返回值数据类型>[DETERMINISTIC] [PIPELINED] AS|IS BEGIN <实现体> END [函数名]
<方法继承属性> ::= <重载属性> | <final 属性> | <重载属性> <final 属性>
<重载属性> ::= [NOT] OVERRDING
<final 属性> ::= FINAL | NOT FINAL | INSTANTIABLE | NOT INSTANTIABLE
<初始化代码> ::= [[<说明部分>]BEGIN<执行部分>[<异常处理部分>]]
<说明部分> ::=[DECLARE]<说明定义>{<说明定义>}
<说明定义>::=<变量说明>|<异常变量说明>|<游标定义>|<子过程定义>|<子函数定义>
<变量说明>::=<变量名>{,<变量名>}<变量类型>[DEFAULT|ASSIGN|:=<表达式>];
<变量类型>::=<DMSQL 程序类型>|< [模式名.]表名.列名%TYPE>|<[模式名.]表名%ROWTYPE>|<记录类型>
<记录类型>::= RECORD(<变量名> <DMSQL 程序类型>{,<变量名> <DMSQL 程序类型>})
<异常变量说明>::=<异常变量名>EXCEPTION[FOR<错误号>]
<游标定义>::=CURSOR <游标名> [FOR<查询表达式>|<表连接>]
<子过程定义>::=PROCEDURE<过程名>[(<参数列>)]<IS|AS><模块体>
<子函数定义>::=FUNCTION<函数名>[(<参数列>)]RETURN<返回数据类型>[PIPELINED]<IS|AS><模块体>
<执行部分>::=<SQL 过程语句序列>{;< SQL 过程语句序列>}
< SQL 过程语句序列>::=[<标号说明>]<SQL 过程语句>;
<标号说明>::=<<<标号名>>>
<SQL 过程语句>::=<SQL 语句>|<SQL 控制语句>
<异常处理部分>::=EXCEPTION<异常处理语句>{;<异常处理语句>};
<异常处理语句>::= WHEN <异常名> THEN < SQL 过程语句序列>
```

图例



使用说明

1. 类声明中定义的对象对于类体而言都是可见的，不需要声明就可以直接引用。这些对象包括变量、游标、异常定义和类型定义；

2. 类体中的过程、函数定义必须和类声明中的声明完全相同。包括过程的名字、参数定义列表的参数名和数据类型定义；

3. 类中可以有重名的成员方法，要求其参数定义列表各不相同。系统会根据用户的调用情况进行重载(OVERLOAD)；

4. 声明类与实现类时，对于确定性函数的指定逻辑与包内函数相同。目前不支持类的确定性函数在函数索引中使用。

权限

使用该语句的用户必须是 DBA 或该类对象的拥有者且具有 CREATE CLASS 数据库权限的用户。

完整的类头、类体的创建如下所示：

```
----类头创建
create class mycls
as
type rec_type is  record (c1 int, c2  int); --类型声明
id          int;                        --成员变量
r           rec_type;            --成员变量
function f1(a int, b int) return rec_type; --成员函数
function mycls(id int , r_c1 int, r_c2 int) return mycls;
--用户自定义构造函数
end;
/
----类体创建
create or replace class body mycls
as
function f1(a int, b int) return rec_type
as
begin
    r.c1 = a;
    r.c2 = b;
    return r;
end;
function mycls(id int, r_c1 int, r_c2 int) return mycls
as
begin
    this.id = id;       --可以使用 this.来访问自身的成员
    r.c1 = r_c1;        --this 也可以省略
    r.c2 = r_c2;
    return this;        --使用 return this 返回本对象
end;
end;
/
```

12.1.3 重编译类

重新对类进行编译，如果重新编译失败，则将类置为禁止状态。

重编功能主要用于检验类的正确性。

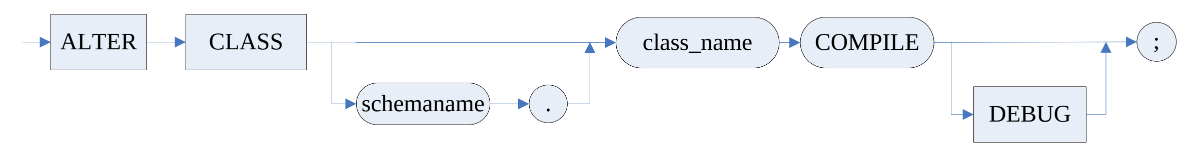
语法格式

```
ALTER CLASS [<模式名>.]<类名> COMPILE [DEBUG];
```

参数

1. <模式名> 指明被重编译的类所属的模式；
2. <类名> 指明被重编译的类的名字；
3. [DEBUG] 可忽略。

图例



权限

执行该操作的用户必须是类的创建者，或者具有 DBA 权限。

举例说明

例 重新编译类

```
ALTER CLASS mycls COMPILE;
```

12.1.4 删除类

类的删除分为两种方式：一是类头的删除，删除类头则会顺带将类体一起删除；另外一种是类体的删除，这种方式只能删除类体，类头依然存在。

12.1.4.1 删除类头

类的删除通过 DROP CLASS 完成，即类头的删除。删除类头的同时会一并删除类体。

语法格式

```
DROP CLASS [<模式名>.]<类名>[RESTRICT | CASCADE];
```

使用说明

1. 如果被删除的类不属于当前模式，必须在语句中指明模式名；
2. 如果一个类的声明被删除，那么对应的类体被自动删除。

权限

执行该操作的用户必须是该类的拥有者，或者具有 DBA 权限。

12.1.4.2 删除类体

从数据库中删除一个类的实现主体对象。

语法格式

```
DROP CLASS BODY [<模式名>.]<类名>[RESTRICT | CASCADE];
```

使用说明

如果被删除的类不属于当前模式，必须在语句中指明模式名。

权限

执行该操作的用户必须是该类的拥有者，或者具有 DBA 权限。

12.1.5 类的使用

类类型同普通的数据类型一样，可以作为表中列的数据类型，DMSQL 程序语句块中变

量的数据类型或过程及函数参数的数据类型。

12.1.5.1 具体使用规则

1．作为表中列类型或其他类成员变量属性的类不能被修改，删除时需要指定 CASCADE 级联删除

类中定义的数据类型，其名称只在类的声明及实现中有效。如果类内的函数的参数或返回值是类内的数据类型，或是进行类内成员变量的复制，需要在 DMSQL 程序中定义一个结构与之相同的类型。

根据类使用方式的不同，对象可分为变量对象及列对象。变量对象指的是在 DMSQL 程序语句块中声明的类类型的变量；列对象指的是在表中类类型的列。变量对象可以修改其属性的值而列对象不能。

2．变量对象的实例化

类的实例化通过 NEW 表达式调用构造函数完成。

3．变量对象的引用

通过'='进行的类类型变量之间的赋值所进行的是对象的引用，并没有复制一个新的对象。

4．变量对象属性访问

可以通过如下方式进行属性的访问。

```
<对象名>.<属性名>
```

5．变量对象成员方法调用

成员方法的调用通过以下方式调用：

```
<对象名>.<成员方法名>(<参数>{,<参数>})
```

如果函数内修改了对象内属性的值，则该修改生效。

6．列对象的插入

列对象的创建是通过 INSERT 语句向表中插入数据完成，插入语句中的值是变量对象，插入后存储在表中的数据即为列对象。

7．列对象的复制

存储在表中的对象不允许对对象中成员变量的修改，通过 into 查询或'='进行的列到变量的赋值所进行的是对象的赋值，生成了一个与列对象数据一样的副本，在该副本上进行的修改不会影响表中列对象的值。

8．列对象的属性访问

通过如下方式进行属性的访问：

```
<列名>.<属性名>
```

9．列对象的方法调用

```
<列名>.<成员方法名>(<参数>{,<参数>})
```

列对象方法调用过程中对类型内属性的修改，都是在列对象的副本上进行的，不会影响列对象的值。

12.1.5.2 应用实例

1．变量对象的应用实例

```
declare
    type ex_rec_t is  record (a int, b  int);  --使用一个同结构的类型代替类定义的类型
    rec ex_rec_t;
    o1  mycls;
    o2   mycls;
begin
    o1 = new mycls(1,2,3);
    o2 = o1;            --对象引用
    rec = o2.r;         --变量对象的成员变量访问
    print rec.a; print rec.b;
    rec = o1.f1(4,5);   --成员函数调用
    print rec.a; print rec.b;
    print  o1.id;   --成员变量访问
end;
```

2．列对象的应用实例

表的创建。

```
Create table tt1(c1 int, c2 mycls);
```

列对象的创建--插入数据。

```
Insert into tt1 values(1, mycls(1,2,3));
```

列对象的复制及访问。

```
Declare
    o mycls;
    id int;
begin
   select top 1 c2 into o from tt1;        --列对象的复制
   select top 1 c2.id into id from tt1;        --列对象成员的访问
end;
```

3．类继承的应用实例

```
CREATE OR REPLACE CLASS cls01 NOT FINAL IS
   name VARCHAR2(10);
   MEMBER FUNCTION get_info RETURN VARCHAR2;
 END;

CREATE OR REPLACE CLASS cls02 UNDER cls01 IS
   ID INT;
   OVERRIDING MEMBER FUNCTION get_info RETURN VARCHAR2;
END;
```

12.2 JAVA CLASS 类型

JAVA 类的定义类似 JAVA 语言语法，类中可定义。

JAVA 类中可以包括以下内容：

1． 类型定义

在类中可以定义游标、异常，可以声明记录类型、数组类型、结构体类型以及内存索引表等数据类型变量。

2. 属性

类中的成员变量，数据类型可以是标准的数据类型，可以是在类外自定义的特殊数据类型。

3. 成员方法

JAVA 类中的成员方法及后文的构造函数包含一个隐含参数，即自身对象，在方法实现中可以通过 this 或 self 来访问自身对象，self 等价于 this。如果不存在重名问题，也可以直接使用对象的属性和方法。

4. 构造函数

构造函数是类内定义及实现的一种特殊的函数，这类函数用于实例化类的对象，构造函数满足以下条件：

1） 函数名和类名相同；

2） 函数没有返回值类型。

构造函数存在以下的约束：

1） 系统为每个类提供两个默认的构造函数，分别为 0 参的构造函数和全参的构造函数；

2） 0 参构造函数的参数个数为 0，实例的对象内所有的属性初始化值为 NULL；

3） 全参构造函数的参数个数及类型和类内属性的个数及属性相同，按照属性的顺序依次读取参数的值并给属性赋值；

4） 用户可以自定义构造函数，一个类可以有多个构造函数，但每个构造函数的参数个数必须不同；

5） 如果用户自定义了 0 个参数、或参数个数同属性个数相同的构造函数，则会覆盖相应的默认构造函数。

12.2.1 定义 JAVA 类

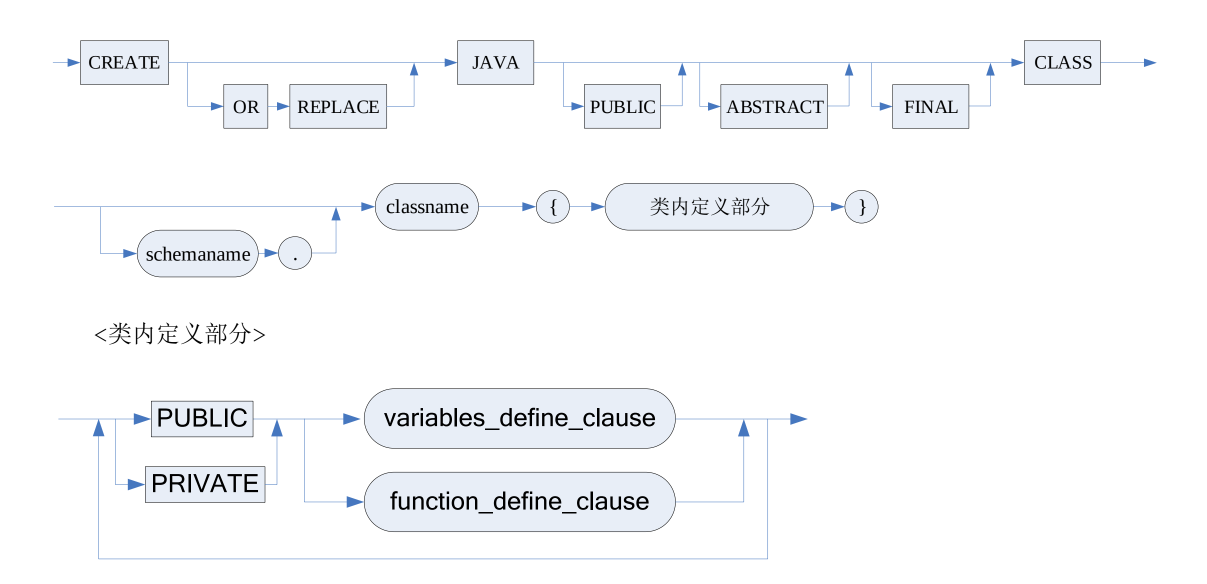
定义通过 CREATE JAVA CLASS 语句来完成，其语法为：

语法格式

```
CREATE [OR REPLACE] JAVA [PUBLIC] [ABSTRACT] [FINAL] CLASS <类名> [EXTENDS [<模式名>.]<父类名>] {<类内定义部分> }
<类内定义部分> :: = <类内定义列表>
<类内定义列表> ::= <类内定义>;{<类内定义>;}

<类内定义> ::= [PUBLIC|PRIVATE] <变量定义>|<方法定义>
<变量定义> ::= <变量属性> <数据类型><变量名列表> [默认值定义]
<变量属性> :: = [STATIC] <final 属性>
<方法定义> ::= [PUBLIC|PRIVATE] [<方法继承属性>] [STATIC]  <返回类型> <函数名><参数列表> { <实现体> }
<方法继承属性> ::= <重载属性> | <FINAL 属性> | <ABSTRACT 属性>
<ABSTRACT 属性> ::= ABSTRACT
<FINAL 属性> ::= FINAL
<重载属性> ::= OVERRIDE
```

图例



使用说明

1. 类中元素可以以任意顺序出现，其中的对象必须在引用之前被声明。
2. 支持对象静态方法声明与调用。可以在方法前添加 static 保留字，以此指明方法为静态方法。静态方法只能以对象名为前缀调用，而不能在对象实例中调用。
3. 支持对象成员方法声明与调用。非 STATIC 类型的非构造函数方法默认为成员方法。成员方法调用时，需要先实例化，实例化参数值缺省为 null。
4. 变量定义还包括游标、异常定义。
5. 方法属性是 PUBLIC，则访问类时可以访问，如果是 PRIVATE 属性，则访问类时不可以访问该方法。
6. 关于 JAVA 类继承，有以下使用限制：
    1) JAVA CLASS 定义默认可继承，FINAL 表示该类不能被继承；
    2) 定义子类时必须指定 EXTENDS 选项；
    3) ABSTRACT 对象不能为 FINAL；
    4) ABSTRACT 对象不能实例化，但是可以用其子类赋值；
    5) 子类对象实例化时，必须对父类和子类的成员变量都赋值，且从父类到子类逐个赋值；
    6) 不支持对象的循环继承；
    7) 不支持对象的多继承，即一个类只能有一个父类；
    8) 不支持父类和子类包含同名变量；
    9) 父类和子类可以同名同参，此时子类必须指定 OVERRIDE；
    10) 方法默认为 NOT OVERRIDING，OVERRIDING 不能与 static 一起使用；
    11) 父类和子类支持同名不同参（参数个数不同、参数个数相同但类型不同）的方法；
    12) 同名且参数个数相同但类型不同时，根据参数类型选择使用的方法；
    13) 方法如果声明为 ABSTRACT，则不能与 FINAL、STATIC 一起使用；
    14) 如果父类有多个 ABSTRACT 方法，子类可以只部分重写，但此时子类必须定义为 ABSTRACT；
    15) ABSTRACT 方法不能具有主体；
    16) 方法默认为可继承，如果声明为 FINAL，则不能被子类重写；
    17) 子类可以赋值给父类；

18）如果父类对应的实例是子类或者子类的孩子，则该父类可以赋值给子类；
19）可以用 ABSTRACT 子类对非 ABSTRACT 父类进行赋值；
20）子类实例赋值给父类后，调用时使用的是父类方法而不是子类方法；
21）支持使用 super 无参方法转换为父类引用；
22）支持使用 this()调用该类构造函数，super()调用父类构造函数；
23）子类必须有新增成员或方法，不能完全为空。

12.2.2 重编译 JAVA 类

重新对 JAVA 类进行编译，如果重新编译失败，则将 JAVA 类置为禁止状态。
重编功能主要用于检验 JAVA 类的正确性。

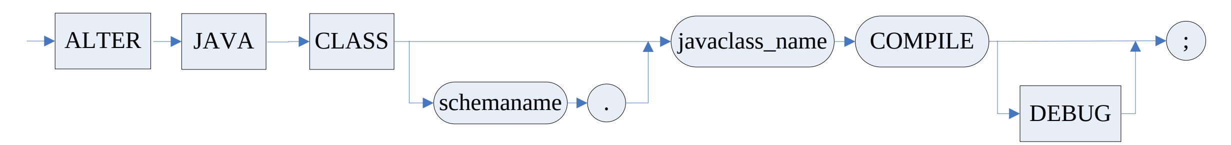
语法格式

```
ALTER JAVA CLASS [<模式名>.]<JAVA 类名> COMPILE [DEBUG];
```

参数

1. <模式名> 指明被重编译的 JAVA 类所属的模式；
2. <JAVA 类名> 指明被重编译的 JAVA 类的名字；
3. [DEBUG] 可忽略。

图例



权限

执行该操作的用户必须是 JAVA 类的创建者，或者具有 DBA 权限。

12.2.3 删除 JAVA 类

JAVA 类的删除通过 DROP CLASS 完成。

语法格式

```
DROP CLASS <类名>[RESTRICT | CASCADE];
```

12.2.4 类的使用

下面列举一个简单的应用实例。在列对象上如何使用 JAVA CLASS。

1.创建 JAVA CLASS。

```
create or replace java class jcls
{
 int a;
  public static int testAdd2(int a, int b) {     //此处创建的是静态 STATIC 方法
         return a + b;
    }
```

```
    public int testAdd3(int a, int b, int c) {     //此处创建的是成员方法
        return a + b +c;
    }
}
```

2．在列对象中使用 JAVA CLASS。

```
create table tt2(c1 int, c2 jcls);
insert into tt2 values(jcls.testadd2(1,2),jcls(1));    //静态方法调用
insert into tt2 values(jcls().testadd3(1,2,3),jcls(2));    //成员方法调用之前必须实例化
```

第 13 章自定义类型

用户使用 CREATE TYPE 语句可以定义记录类型、对象类型、命名的数组类型、集合类型等，如果在对象类型中声明了过程或方法，可以使用 CREATE TYPE BODY 定义这些过程和方法。

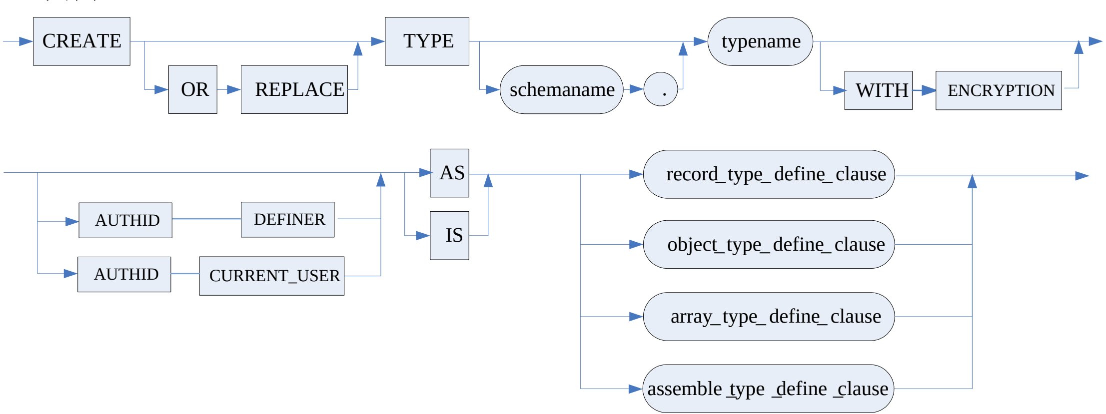
13.1 创建类型

可以使用 CREATE TYPE 语句创建记录类型、对象类型、数组和集合类型。

语法格式

```
CREATE [OR REPLACE] TYPE [<模式名>.]<类型名>[WITH ENCRYPTION] [<调用权限子句>] AS|IS <记录类型定义子句>|<对象类型定义子句>|<数组类型定义子句>|<集合类型定义子句>
[<调用权限子句>]::= AUTHID DEFINER
                  | AUTHID CURRENT_USER
<对象类型定义子句> ::= OBJECT [UNDER [<模式名>.]<父类型名>] (<对象定义>,{<对象定义>})[[NOT] FINAL] [[NOT] INSTANTIABLE]
<对象定义> ::= <变量列表定义>|<过程声明>|<函数声明>|<构造函数声明>
<过程声明> ::= [<方法继承属性>][STATIC|MEMBER] PROCEDURE <过程名> <参数列表>
<函数声明> ::= [<方法继承属性>][MAP] [STATIC|MEMBER] FUNCTION <函数名> <参数列表> RETURN <返回值数据类型>[DETERMINISTIC][PIPELINED]
<方法继承属性> ::= <重载属性> | <final 属性> | <重载属性> <final 属性>
<重载属性> ::= [NOT] OVERRDING
<final 属性> ::= FINAL | NOT FINAL | INSTANTIABLE | NOT INSTANTIABLE
<构造函数声明> ::= CONSTRUCTOR FUNCTION <函数名> <参数列表> RETURN SELF AS RESULT
<记录类型定义子句> ::= RECORD(变量列表定义)
<数组类型定义子句> ::= ARRAY <数据类型>' [' [<常量表达式>]{,[<常量表达式>]}']'
<集合类型定义子句> ::= <数组集合定义子句>|<嵌套表定义子句>|<索引表定义子句>
<数组集合定义子句> ::= VARRAY(<常量表达式>) OF <数据类型>
<嵌套表定义子句> ::= TABLE OF <数据类型>
<索引表定义子句> ::= TABLE OF <数据类型> [INDEX BY <数据类型>]
```

图例



使用说明

1. 对象类型中过程和函数的声明都是前向声明，类型定义中不包括任何实现代码；达梦系统中对象类型与类是等价的，关于类的说明详见第 12 章；
2. 对象类型中过程和函数可以声明为 STATIC 类型，表明为静态过程或函数；也可以声明为 MEMBER，表明为成员过程或函数，非 STATIC 且非构造函数的方法缺省为成员方法。MAP 表示将对象类型的实例映射为标量数值，只能用于成员函数；
3. 关于对象类型的继承，参考 12.1 节中类继承的相关说明；
4. WITH ENCRYPTION 选项，指定是否对自定义类型定义进行加密；
5. 记录类型的定义格式与对象类型类似，但记录类型中不能有过程和函数声明；
6. 在<数组类型定义子句>的数组长度定义的[]内添加',' 可以定义多维数组。若指定了常量表达式，则定义的是静态数组，其数组长度是固定的。若没有指定常量表达式，则定义的是动态数组，其数组长度是在使用时指定。理论上 DM 支持静态数组的每一个维度的最大长度为 65534，动态数组的每一个维度的最大长度为 2147483646，但是数组最大长度同时受系统内部空间大小的限制，如果超出堆栈/堆的空间限制，系统会报错。
7. 数组集合类型中的常量表达式定义了其最大容量，其数组元素数据类型可以是基础类型，也可以是自定义数据类型。
8. 嵌套表类型和索引表类型没有元素个数限制，元素数据类型可以是基础数据类型也可以是其它自定义类型或是对象、记录、静态数组，但是不能是动态数组；第二个则是索引表的下标类型，目前仅支持 INTEGER/INT 和 VARCHAR 两种类型，分别代表整数下标和字符串下标。对于 VARCHAR 类型，长度不能超过 1024。

权限

1. 使用该语句的用户必须是 DBA 或具有 CREATE TYPE 数据库权限的用户。
2. 可以用关键字 AUTHID DEFINER |AUTHID CURRENT_USER 指定自定义类型的调用者权限，若为 DEFINER，则采用自定义类型定义者权限，若为 CURRENT_USER 则为当前用户权限，默认为定义者权限。

13.2 创建类型体

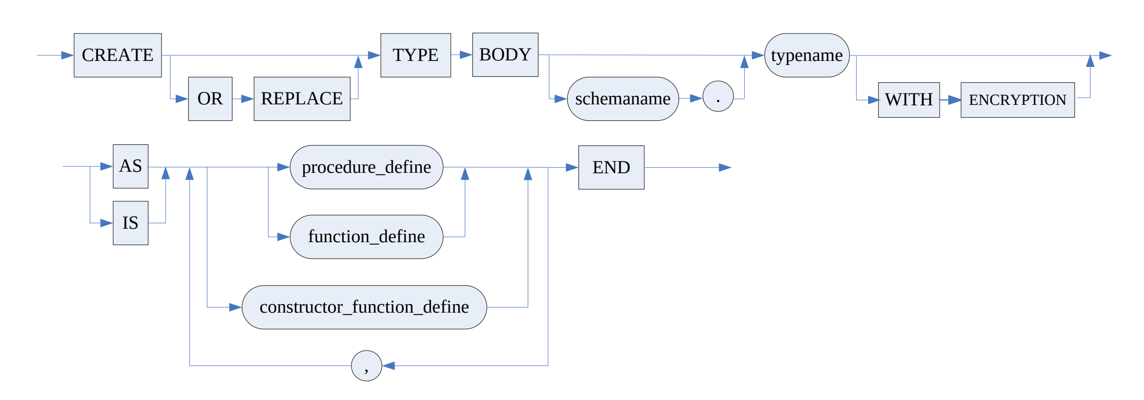
对于对象类型中声明的过程和函数，在类型体中进行实现。

语法格式

```
CREATE [OR REPLACE] TYPE BODY [<模式名>.]<类型名>[WITH ENCRYPTION] AS|IS <对象类型体定义子句> END
<对象类型体定义子句>::= <对象类型体定义>,{<对象类型体定义>}
<对象类型体定义>::= <过程实现>|<函数实现>|<构造函数实现>
<过程实现> ::= [<方法继承属性>][STATIC|MEMBER] PROCEDURE <过程名> <参数列表> AS|IS BEGIN <实现体> END [过程名]
<函数实现> ::= [<方法继承属性>][MAP] [STATIC|MEMBER] FUNCTION <函数名><参数列表> RETURN <返回值数据类型>[DETERMINISTIC] [PIPELINED] AS|IS BEGIN <实现体> END [函数名]
<方法继承属性> ::= <重载属性> | <final 属性> | <重载属性> <final 属性>
<重载属性> ::= [NOT] OVERRDING
<final 属性> ::= FINAL | NOT FINAL | INSTANTIABLE | NOT INSTANTIABLE
```

```
    <构造函数实现> ::= CONSTRUCTOR FUNCTION <函数名> <参数列表>  RETURN SELF AS RESULT AS|IS BEGIN <实现体> END [函数名]
```

图例



使用说明

1. 对象类型体中的过程、函数定义必须和类型定义中的前向声明完全相同。包括过程的名字、参数定义列表的参数名和数据类型定义；

权限

使用该语句的用户必须是 DBA 或该类型对象的拥有者且具有 CREATE TYPE 数据库权限的用户。

13.3 重编译类型

重新对类型进行编译，如果重新编译失败，则将类型置为禁止状态。

重编功能主要用于检验类型的正确性。

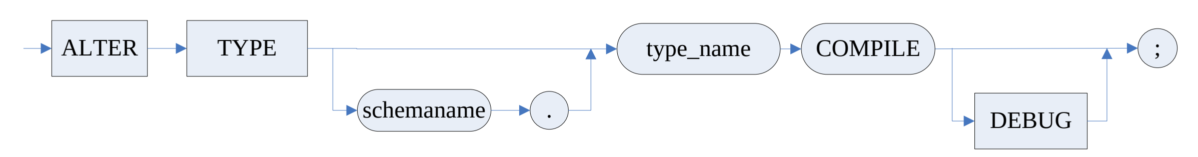
语法格式

```
ALTER TYPE [<模式名>.]<类型名> COMPILE [DEBUG];
```

参数

1. <模式名>　　指明被重编译的类型所属的模式；
2. <类型名>　　指明被重编译的类型的名字；
3. [DEBUG]　　可忽略。

图例



权限

执行该操作的用户必须是类型的创建者，或者具有 DBA 权限。

13.4 删除类型

类型的删除分为类型删除和类型体的删除。对于拥有类型体的对象类型，删除类型会将类型体一起删除；删除类型体的话，类型本身依然存在。

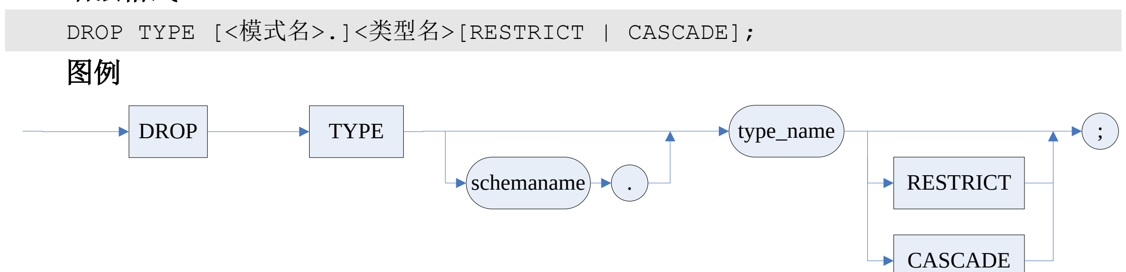
13.4.1 删除类型

使用 DROP TYPE 完成类型的删除。对于拥有类型体的对象类型，删除类型会将类型体一起删除。

语法格式

```
DROP TYPE [<模式名>.]<类型名>[RESTRICT | CASCADE];
```

图例



使用说明

1. 如果被删除的类型不属于当前模式，必须在语句中指明模式名；
2. 如果一个拥有类型体的对象类型被删除，那么对应的类型体被自动删除。

权限

执行该操作的用户必须是该类型的拥有者，或者具有 DBA 权限。

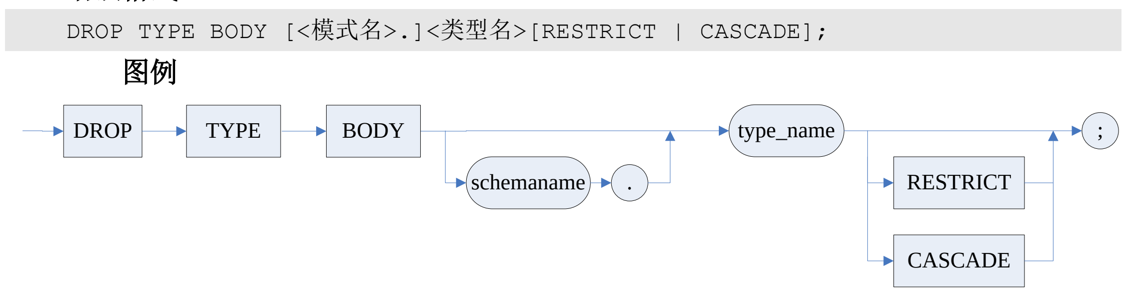
13.4.2 删除类型体

使用 DROP TYPE BODY 删除一个对象类型的类型体。

语法格式

```
DROP TYPE BODY [<模式名>.]<类型名>[RESTRICT | CASCADE];
```

图例



使用说明

如果被删除的类型体不属于当前模式，必须在语句中指明模式名。

权限

执行该操作的用户必须是该类型的拥有者，或者具有 DBA 权限。

13.5 自定义类型的使用

13.5.1 使用规则

1. 对象类型与类等价，类的使用规则可详见第 12 章《类类型》 ；
2. 创建的记录类型、数组类型和集合类型，可以直接在 DMSQL 程序语句块中使用，不必在语句块中声明类型，使用方式可参见 10.1.1 节部分；

3. 用户自定义数据类型可以作为其他用户自定义数据类型的元素类型或成员变量类型；
4. 只有对象类型可以直接作为表中列的数据类型；其他类型只能作为对象类型中成员变量的类型或类型中嵌套使用的数据类型。但含有索引表类型和游标类型的对象类型也不能作为表中列的数据类型。

13.5.2 应用实例

创建一个用来表示复数的对象类型，有实数部分和虚数部分，并实现了复数的加与减的操作。

```
CREATE TYPE COMPLEX AS OBJECT(
   RPART    REAL,
   IPART    REAL,
   FUNCTION PLUS(X COMPLEX) RETURN COMPLEX,
   FUNCTION LES(X COMPLEX) RETURN COMPLEX
);
/

CREATE TYPE BODY COMPLEX AS
   FUNCTION PLUS(X COMPLEX) RETURN COMPLEX IS
   BEGIN
   RETURN COMPLEX(RPART+X.RPART, IPART+X.IPART);
   END;

   FUNCTION LES(X COMPLEX) RETURN COMPLEX IS
   BEGIN
    RETURN COMPLEX(RPART-X.RPART, IPART-X.IPART);
   END;
END;
```

建立表 c_tab，表中的第二列的列类型为 complex 对象类型。

```
CREATE TABLE C_TAB(C1 INT, C2 COMPLEX);
```

向表 c_tab 中插入数据。

```
INSERT INTO C_TAB VALUES(1, COMPLEX(2,3));
INSERT INTO C_TAB VALUES(2, COMPLEX(4,2).PLUS(COMPLEX(2,3)));
```

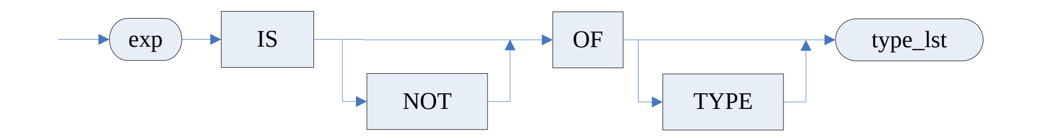
13.5.3 IS OF TYPE 的使用

IS OF TYPE 谓词用于判断一个表达式对应实例是否是指定 type 列表的子类或者是完全相同的类型。

语法格式

```
<表达式> IS [NOT] OF [TYPE] (<TYPE 列表>)
```

图例



使用说明

1. 用户需要具有表达式对应 TYPE 和 TYPE 列表中涉及所有 TYPE 的执行权限。

举例说明

```
CREATE OR REPLACE TYPE TYPE01 AS OBJECT(
NAME VARCHAR2(10))
NOT FINAL;
/
CREATE TABLE T1(C1 INT , C2 TYPE01);
INSERT INTO T1 VALUES(1, TYPE01('WSY'));
COMMIT;

SELECT * FROM T1 WHERE C1 IS OF (TYPE01);
```

第 14 章 触发器

DM 是一个具有主动特征的数据库管理系统，其主动特征包括约束机制和触发器机制。通过触发器机制，用户可以定义、删除和修改触发器。DM 自动管理和运行这些触发器，从而体现系统的主动性，方便用户使用。

触发器(TRIGGER)定义为当某些与数据库有关的事件发生时，数据库应该采取的操作。这些事件包括全局对象、数据库下某个模式、模式下某个基表上的 INSERT、DELETE 和 UPDATE 操作。触发器与存储模块类似，都是在服务器上保存并执行的一段 DMSQL 程序语句。不同的是：存储模块必须被显式地调用执行，而触发器是在相关的事件发生时由服务器自动地隐式地激发。触发器是激发它们的语句的一个组成部分，即直到一个语句激发的所有触发器执行完成之后该语句才结束，而其中任何一个触发器执行的失败都将导致该语句的失败，触发器所做的任何工作都属于激发该触发器的语句。

触发器为用户提供了一种自己扩展数据库功能的方法。关于触发器应用的例子有：

1. 利用触发器实现表约束机制(如：PRIMARY KEY、FOREIGN KEY、CHECK 等)无法实现的复杂的引用完整性；
2. 利用触发器实现复杂的事务规则(如：想确保薪水增加量不超过 25%)；
3. 利用触发器维护复杂的缺省值(如：条件缺省)；
4. 利用触发器实现复杂的审计功能；
5. 利用触发器防止非法的操作。

触发器是应用程序分割技术的一个基本组成部分，它将事务规则从应用程序的代码中移到数据库中，从而可确保加强这些事务规则并提高它们的性能。触发器中可以定义变量，但是必须以 DECLARE 开头。

需要说明的是，在 DM 的数据守护环境下，备库上定义的触发器是不会被触发的。

在本章各例中，如不特别说明，各例均使用示例库 BOOKSHOP，用户均为建表者 SYSDBA。

14.1 触发器的定义

触发器分为表触发器、事件触发器和时间触发器。表触发器是对表里数据操作引发的数据库的触发；事件触发器是对数据库对象操作引起的数据库的触发；时间触发器是一种特殊的事件触发器。

14.1.1 表触发器

14.1.1.1 表触发器语法

用户可使用触发器定义语句(CREATE TRIGGER)在一张基表上创建触发器。下面是表触发器定义语句的语法。

语法格式

```
CREATE [OR REPLACE] TRIGGER [<模式名>.]<触发器名> [WITH  ENCRYPTION]
```

```
<触发限制描述> [REFERENCING <trig_referencing_list>][FOR EACH {ROW | STATEMENT}][WHEN (<条件表达式>)]<触发器体>
<trig_referencing_list>::= <referencing_1>|<referencing_2>
<referencing_1>::=OLD [ROW] [AS] <引用变量名> [ NEW [ROW] [AS] <引用变量名>]
<referencing_2>::=NEW [ROW] [AS] <引用变量名> [ OLD [ROW] [AS] <引用变量名>]
<触发限制描述>::=<触发限制描述 1> | <触发限制描述 2>
<触发限制描述 1>::= <BEFORE|AFTER> <触发事件列表> [LOCAL] ON <触发表名>
<触发限制描述 2>::= INSTEAD OF <触发事件列表> [LOCAL] ON <触发视图名>
<触发表名>::=[<模式名>.]<基表名>
<触发事件>::=INSERT|DELETE|{UPDATE|{UPDATE OF<触发列清单>}}
<触发事件列表>::=<触发事件> | {<触发事件列表> OR <触发事件>}
```

参数

1. <触发器名> 指明被创建的触发器的名称；

2. BEFORE 指明触发器在执行触发语句之前激发；

3. AFTER 指明触发器在执行触发语句之后激发；

4. INSTEAD OF 指明触发器执行时替换原始操作；

5. <触发事件> 指明激发触发器的事件。INSTEAD OF 中不支持 {UPDATE OF <触发列清单>}；

6. <基表名>指明被创建触发器的基表的名称；

7. WITH ENCRYPTION 选项，指定是否对触发器定义进行加密；

8. REFERENCING 子句 指明相关名称可以在元组级触发器的触发器体和 WHEN 子句中利用相关名称来访问当前行的新值或旧值，缺省的相关名称为 OLD 和 NEW；

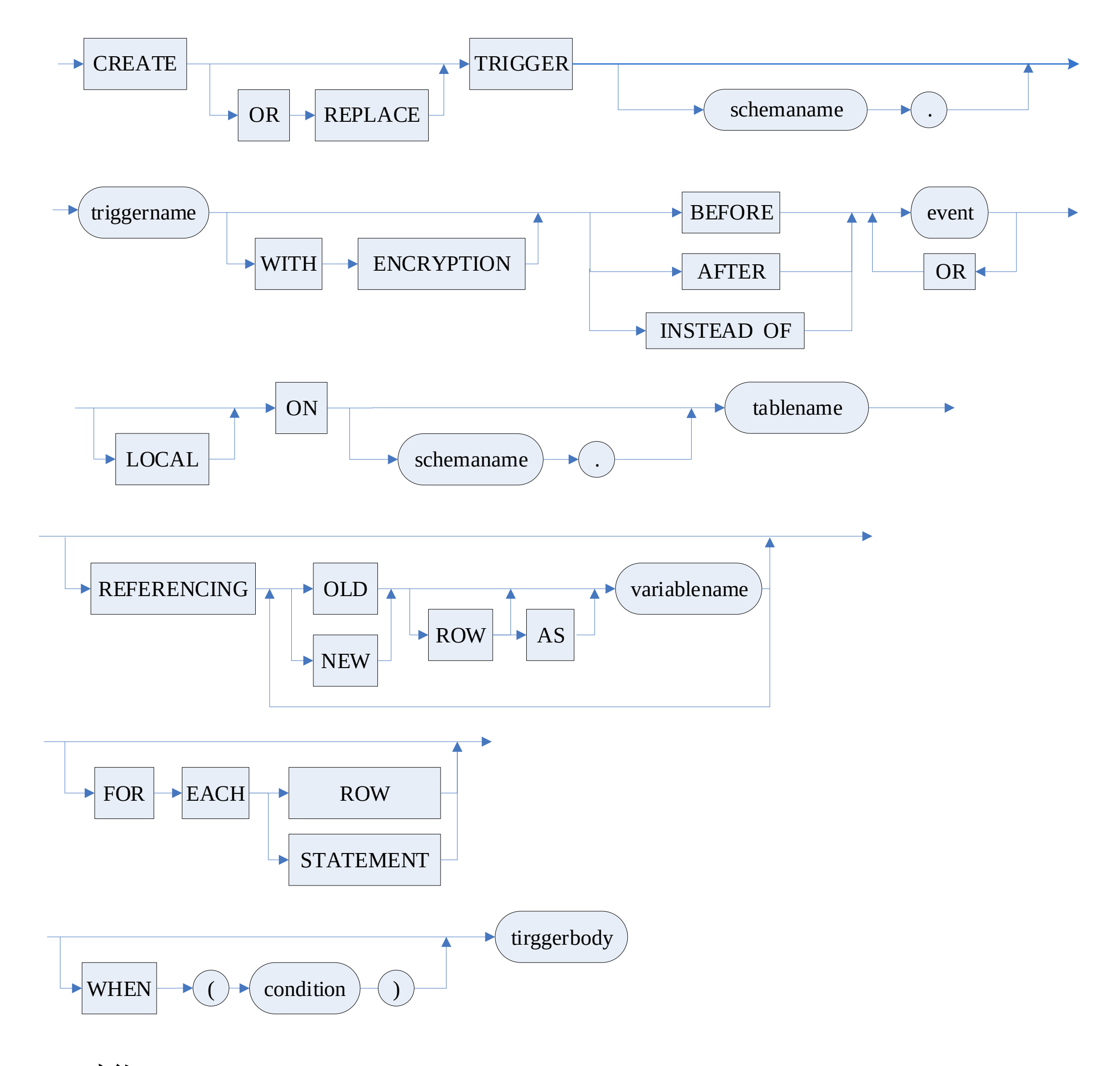
9. <引用变量名> 标识符，指明行的新值或旧值的相关名称；

10. FOR EACH 子句 指明触发器为元组级或语句级触发器。FOR EACH ROW 表示为元组级触发器，它受被触发命令影响、且 WHEN 子句的表达式计算为真的每条记录激发一次。FOR EACH STATEMENT 为语句级触发器，它对每个触发命令执行一次。FOR EACH 子句缺省则为语句级触发器；

11. WHEN 子句 只允许为元组级触发器指定 WHEN 子句，它包含一个布尔表达式，当表达式的值为 TRUE 时，执行触发器；否则，跳过该触发器；

12. <触发器体> 触发器被触发时执行的 SQL 过程语句块。

图例



功能

创建触发器，并使其处于允许状态。

使用说明

1．<触发器名>是触发器的名称，它不能与模式内的其他模式级对象同名；

2．可以使用 OR REPLACE 选项来替换一个触发器，但是要注意被替换的触发器的触发表不能改变。如果要在同一模式内不同的表上重新创建一个同名的触发器，则必须先删除该触发器，然后再创建；

3．<触发事件子句>说明激发触发器的事件；<触发器体>是触发器的执行代码；<引用子句>用来引用正处于修改状态下的行中的数据。如果指定了<触发条件>子句，则首先对该条件表达式求值，<触发器体>只有在该条件为真值时才运行。<触发器体>是一个 DMSQL 程序语句块，它与存储模块定义语句中<模块体>的语法基本相同；

4．在一张基表上允许创建的表触发器的个数没有限制，一共允许有 12 种类型。它们分别是：BEFORE INSERT 行级、BEFORE INSERT 语句级、AFTER INSERT 行级、AFTER INSERT 语句级、BEFORE UPDATE 行级、BEFORE UPDATE 语句级、AFTER UPDATE 行级、AFTER UPDATE 语句级、BEFORE DELETE 行级、BEFORE DELETE 语句级、AFTER DELETE 行级和 AFTER DELETE 语句级；

5．触发器是在 DML 语句运行时激发的。执行 DML 语句的算法步骤如下：

1） 如果有语句级前触发器的话，先运行该触发器；

2） 对于受语句影响每一行：

a） 如果有行级前触发器的话，运行该触发器；

b） 执行该语句本身；

c） 如果有行级后触发器的话，运行该触发器。

3） 如果有语句级后触发器的话，运行该触发器。

6．INSTEAD OF 触发器仅允许建立在视图上，并且只支持行级触发；

7．表级触发器不支持跨模式，即<触发器名>必须和<触发表名>、<触发视图名>的模式名一致；

8．水平分区子表、HUGE 表不支持表级触发器；

9．在 MPP 环境下，执行 LOCAL 类型触发器时，会话会被临时变为 LOCAL 类型，因此触发器体只会在本节点执行，不会产生节点间的数据交互，触发器体中只能包含表的值插入操作，如果插入数据的目标节点不是于本节点，则会报错，随机分布表没有此限制。

14.1.1.2 表触发器详解

下面对表级触发器的触发动作、级别和时机进行详细介绍。

14.1.1.2.1 触发动作

激发表级触发器的触发动作是三种数据操作命令，即 INSERT、DELETE 和 UPDATE 操作。在触发器定义语句中用关键字 INSERT、DELETE 和 UPDATE 指明构成一个触发器事件的数据操作的类型，其中 UPDATE 触发器会依赖于所修改的列，在定义中可通过 UPDATE OF <触发列清单>的形式来指定所修改的列，<触发列清单>指定的字段数不能超过 128 个。

在 PERSON.PERSON 上建立触发器。如下例所示：

```
SET SCHEMA PERSON;

CREATE OR REPLACE TRIGGER TRG_UPD
AFTER UPDATE OF NAME,PHONE ON PERSON.PERSON
BEGIN
  PRINT 'UPDATE OPERATION ON COLUMNS NAME OR PHONE OF PERSON';
END;

SET SCHEMA SYSDBA;
```

当对表 PERSON 进行更新操作，并且更新的列中包括 NAME 或 PHONE 时，此例中定义的触发器 TRG_UPD 将被激发。

如果一个触发器的触发事件为 INSERT，则该触发器被称为 INSERT 触发器，同样也可以这样来定义 DELETE 触发器和 UPDATE 触发器。一个触发器的触发事件也可以是多个数据操作命令的组合，这时这个触发器可由多种数据操作命令激发。如下例所示：

```
SET SCHEMA PERSON;

CREATE OR REPLACE TRIGGER TRG_INS_DEL
AFTER INSERT OR DELETE ON PERSON.PERSON
BEGIN
  PRINT 'INSERT OR DELETE OPERATION ON PERSON';
END;

SET SCHEMA SYSDBA;
```

此例中的触发器 TRG_INS_DEL 既是 INSERT 触发器又是 DELETE 触发器，对基表 T1 的 INSERT 和 DELETE 操作都会激发该触发器。

14.1.1.2.2 触发级别

根据触发器的级别可分为元组级 (也称行级)和语句级。

元组级触发器，对触发命令所影响的每一条记录都激发一次。假如一个 DELETE 命令从表中删除了 1000 行记录，那么这个表上的元组级 DELETE 触发器将被执行 1000 次。元组级触发器常用于数据审计、完整性检查等应用中。元组级触发器是在触发器定义语句中通过 FOR EACH ROW 子句创建的。对于元组级触发器，可以用一个 WHEN 子句来限制针对当前记录是否执行该触发器。WHEN 子句包含一条布尔表达式，当它的值为 TRUE 时，执行触发器；否则，跳过该触发器。

语句级触发器，对每个触发命令执行一次。例如，对于一条将 500 行记录插入表 TABLE_1 中的 INSERT 语句，这个表上的语句级 INSERT 触发器只执行一次。语句级触发器一般用于对表上执行的操作类型引入附加的安全措施。语句级触发器是在触发器定义语句中通过 FOR EACH STATEMENT 子句创建的，该子句可缺省。

以下分别是元组级触发器和语句级触发器的例子。

```
SET SCHEMA PERSON;

CREATE OR REPLACE TRIGGER TRG_DEL_ROW
BEFORE DELETE ON PERSON.PERSON
FOR EACH ROW          -- 元组级：此子句一定不能省略
BEGIN
  PRINT 'DELETE' || :OLD.NAME|| ' ON PERSON';
END;

CREATE OR REPLACE TRIGGER TRG_INS_ST
AFTER INSERT ON PERSON.PERSON
FOR EACH STATEMENT    -- 语句级：此子句可省略
BEGIN
  PRINT 'AFTER INSERT ON PERSON';
END;

SET SCHEMA SYSDBA;
```

14.1.1.2.3 触发时机

触发时机通过两种方式指定。一是通过指定 BEFORE 或 AFTER 关键字，选择在触发动作之前或之后运行触发器；二是通过指定 INSTEAD OF 关键字，选择在动作触发的时候，替换原始操作，INSTEAD OF 允许建立在视图上，并且只支持行级触发。

在元组级触发器中可以引用当前修改的记录在修改前后的值，修改前的值称为旧值，修改后的值称为新值。对于插入操作不存在旧值，而对于删除操作则不存在新值。

对于新、旧值的访问请求常常决定一个触发器是 BEFORE 类型还是 AFTER 类型。如果需要通过触发器对插入的行设置列值，那么为了能设置新值，需要使用一个 BEFORE 触发器，因为在 AFTER 触发器中不允许用户设置已插入的值。在审计应用中则经常使用 AFTER

触发器，因为元组修改成功后才有必要运行触发器，而成功地完成修改意味着成功地通过了该表的引用完整性约束。

例 1 BEFORE 触发器和 AFTER 触发器的举例。

BEFORE 触发器示例

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER TRG_INS_BEFORE
BEFORE INSERT ON OTHER.READER
FOR EACH ROW
BEGIN
  :NEW.READER_ID:=:NEW.READER_ID+1;
END;

SET SCHEMA SYSDBA;
```

该触发器在插入一条记录前，将记录中 COL1 列的值加 1。

```
CREATE TABLE T_TEMP(C1 INT,C2 CHAR(20));
```

新建表 T_TEMP。

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER TRG_INS_AFTER
AFTER INSERT ON OTHER.READER
FOR EACH ROW
BEGIN
  INSERT INTO SYSDBA.T_TEMP VALUES(:NEW.READER_ID, 'INSERT ON READER');
END;

SET SCHEMA SYSDBA;
```

该触发器在插入一条记录后，将插入的值以及操作类型记录到用于审计的表 T_TEMP 中。

例 2 INSTEAD OF 触发器举例。

```
create table t1(a int,b int);
insert into t1 values(10,10);
insert into t1 values(11,11);
create view v1 as select * from t1;
```

在视图 v1 上创建 INSTEAD OF 触发器。

```
CREATE OR REPLACE TRIGGER tri1
INSTEAD OF UPDATE ON v1
BEGIN
  insert into t1 values(111,111);  ----替换动作
END;
```

当执行 UPDATE 动作时候，就会触发。将下面的动作替换成触发器里的动作。

```
update v1 set a=100 where a=10;
```

查询结果：

```
A   B
```

```
10  10
11  11
111 111
```

由上面的查询结果可以看出。更新操作并没有成功，而是被触发器中的替换动作替换了。这就是 INSTEAD OF 的妙用之处。

14.1.1.2.4 总结

综上所述，在一张基表上所允许的可能的合法表级触发器类型共有 12 种，如表 14.1.1 所示。

表 14.1.1 表触发器类型

名　　称	功　　能
BEFORE INSERT	在一个 INSERT 处理前激发一次
AFTER INSERT	在一个 INSERT 处理后激发一次
BEFORE DELETE	在一个 DELETE 处理前激发一次
AFTER DELETE	在一个 DELETE 处理后激发一次
BEFORE UPDATE	在一个 UPDATE 处理前激发一次
AFTER UPDATE	在一个 UPDATE 处理后激发一次
BEFORE INSERT FOR EACH ROW	每条新记录插入前激发
AFTER INSERT FOR EACH ROW	每条新记录插入后激发
BEFORE DELETE FOR EACH ROW	每条记录被删除前激发
AFTER DELETE FOR EACH ROW	每条记录被删除后激发
BEFORE UPDATE FOR EACH ROW	每条记录被修改前激发
AFTER UPDATE FOR EACH ROW	每条记录被修改后激发

14.1.1.3 触发器激发顺序

下面是执行 DML 语句的算法步骤：

1. 如果有语句级前触发器的话，先运行该触发器；
2. 对于受语句影响每一行：
    1） 如果有行级前触发器的话，运行该触发器；
    2） 执行该语句本身；
    3） 如果有行级后触发器的话，运行该触发器。
3. 如果有语句级后触发器的话，运行该触发器。

为了说明上面的算法，假设我们用 OTHER.READER 表为例，并在其上创建了所有四种 UPDATE 触发器，即之前、之后、行级前和行级后。其代码如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Reader_Before_St
BEFORE UPDATE ON OTHER.READER
BEGIN
  PRINT 'BEFORE UPDATE TRIGGER FIRED';
END;

CREATE OR REPLACE TRIGGER Reader_After_St
```

```
AFTER UPDATE ON OTHER.READER
BEGIN
  PRINT 'AFTER UPDATE TRIGGER FIRED';
END;

CREATE OR REPLACE TRIGGER  Reader_Before_Row
BEFORE UPDATE ON OTHER.READER
FOR EACH ROW
BEGIN
  PRINT 'BEFORE UPDATE EACH ROW TRIGGER FIRED';
END;

CREATE OR REPLACE TRIGGER  Reader_After_Row
AFTER UPDATE ON OTHER.READER
FOR EACH ROW
BEGIN
  PRINT 'AFTER UPDATE EACH ROW TRIGGER FIRED';
END;


SET SCHEMA SYSDBA;
```

现在，执行更新语句：

```
UPDATE OTHER.READER SET AGE=AGE+1;
```

该语句对三行有影响。语句级前触发器和语句级后触发器将各自运行一次，而行级前触发器和行级后触发器则各运行三次。因此，服务器返回的打印消息应为：

```
BEFORE UPDATE TRIGGER FIRED
BEFORE UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE EACH ROW TRIGGER FIRED
BEFORE UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE EACH ROW TRIGGER FIRED
BEFORE UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE TRIGGER FIRED
```

同类触发器的激发顺序没有明确的定义。如果顺序非常重要的话，应该把所有的操作组合在一个触发器中。

14.1.1.4 新、旧行值的引用

前面曾经提到，在元组级触发器内部，可以访问正在处理中的记录的数据，这种访问是通过两个引用变量:OLD 和:NEW 实现的。:OLD 表示记录被处理前的值，:NEW 表示记录被处理后的值，标识符前面的冒号说明它们是宿主变量意义上的连接变量，而不是一般的 DMSQL 程序变量。我们还可以通过引用子句为这两个行值重新命名。

引用变量与其它变量不在同一个命名空间，所以变量可以与引用变量同名。在触发器体中使用引用变量时，必须采用下列形式：

```
:引用变量名.列名
```

其中，列名必须是触发表中存在的列，否则编译器将报错。

下表总结了标识符:OLD 和:NEW 的含义。

表 14.1.2 标识符:OLD 和:NEW 的含义

触发语句	标识符:OLD	标识符:NEW
INSERT	无定义，所有字段都为 NULL	该语句结束时将插入的值
UPDATE	更新前行的旧值	该语句结束时将更新的值
DELETE	行删除前的旧值	无定义，所有字段都为 NULL

:OLD 引用变量只能读取，不能赋值(因为设置这个值是没有任何意义的)；而:NEW 引用变量则既可读取，又可赋值(当然必须在 BEFORE 类型的触发器中，因为数据操作完成后再设置这个值也是没有意义的)。通过修改:NEW 引用变量的值，我们可以影响插入或修改的数据。

:NEW 行中使用的字段数不能超过 255 个，:NEW 行与:OLD 行中使用的不同字段总数不能超过 255 个。

注意：对于 INSERT 操作，引用变量:OLD 无意义；而对于 DELETE 操作，引用变量:NEW 无意义。如果在 INSERT 触发器体中引用:OLD，或者在 DELETE 触发器体中引用:NEW，不会产生编译错误。但是在执行时，对于 INSERT 操作，:OLD 引用变量的值为空值；对于 DELETE 操作，:NEW 引用变量的值为空值，且不允许被赋值。

例 1 下例中触发器 GenerateValue 使用了:OLD 引用变量。该触发器是一个 UPDATE 前触发器，其目的是做更新操作时不论是否更新表中某列的值，该列的值保持原值不变。这里使用 PRODUCTION.PRODUCT 表为例。

```
SET SCHEMA PRODUCTION;


CREATE OR REPLACE TRIGGER GenerateValue
BEFORE UPDATE ON PRODUCTION.PRODUCT
FOR EACH ROW
BEGIN
:new.NOWPRICE:=:old.NOWPRICE;
END;


SET SCHEMA SYSDBA;
```

当执行一个 UPDATE 语句时，无论用户是否给定 NOWPRICE 字段的值，触发器都将自动保持原来的 NOWPRICE 值不变。例如，用户查询 PRODUCTID=1 的一行数据的 NOWPRICE 值。

```
SELECT NOWPRICE FROM PRODUCTION.PRODUCT WHERE PRODUCTID=1;
```

可得到结果为 NOWPRICE=15.2000;

用户可以执行如下所示的 UPDATE 语句。

```
UPDATE PRODUCTION.PRODUCT SET NOWPRICE =1.0000 WHERE PRODUCTID=1;
```

再次执行查询语句。

```
SELECT NOWPRICE FROM PRODUCTION.PRODUCT WHERE PRODUCTID=1;
```

可发现结果仍然为 NOWPRICE=15.2000，而非修改的 1.0000。

例 2 下例中触发器 GenerateValue 使用了:NEW 引用变量。该触发器是一个 INSERT 前触发器，其目的是自动生成一些字段值。这里使用 PRODUCTION.PRODUCT 表为例。

```
SET SCHEMA PRODUCTION;


CREATE OR REPLACE TRIGGER GenerateValue
```

```
BEFORE INSERT ON PRODUCTION.PRODUCT
FOR EACH ROW
BEGIN
:new.NOWPRICE:=:new.ORIGINALPRICE*:new.DISCOUNT;
END;

SET SCHEMA SYSDBA;
```

触发器 GenerateValue 实际上是修改引用变量:NEW 的值，这就是:NEW 引用变量的用途之一。当执行一个 INSERT 语句时，无论用户是否给定 NOWPRICE 字段的值，触发器都将自动利用 ORIGINALPRICE 字段和 DISCOUNT 字段来计算 NOWPRICE。例如，用户可以执行如下所示的 INSERT 语句。

```
INSERT INTO PRODUCTION.PRODUCT
(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,
SATETYSTOCKLEVEL,ORIGINALPRICE,DISCOUNT,SELLSTARTTIME)
VALUES('老人与海','海明威','上海出版社','2006-8-1','9787532740088',1,'10','100','7.5','2008-1-10');
```

即使为 NOWPRICE 字段指定了值，该值也会被忽略，因为触发器将改变该值。

```
INSERT INTO PRODUCTION.PRODUCT
(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,
SATETYSTOCKLEVEL,ORIGINALPRICE,DISCOUNT,NOWPRICE,SELLSTARTTIME)
VALUES('老人与海','海明威','上海出版社','2006-8-1','9787532740089',1,'10','100','7.5','88','2008-1-10');
```

新插入元组的 NOWPRICE 字段将被触发器修改为 750.0000，而不是语句中的 88。

14.1.1.5 触发器谓词

如前面介绍的，触发事件可以是多个数据操作的组合，即一个触发器可能既是 INSERT 触发器，又是 DELETE 或 UPDATE 触发器。当一个触发器可以为多个 DML 语句触发时，在这种触发器体内部可以使用三个谓词：INSERTING、DELETING 和 UPDATING 来确定当前执行的是何种操作。这三个谓词的含义如下表所示。

表 14.1.3 触发器谓词

谓 词	状 态
INSERTING	当触发语句为 INSERT 时为真，否则为假
DELETING	当触发语句为 DELETE 时为真，否则为假
UPDATING[(<列名>)]	未指定列名时，当触发语句为 UPDATE 时为真，否则为假；指定某一列名时，当触发语句为对该列的 UPDATE 时为真，否则为假

虽然在其他 DMSQL 程序语句块中也可以使用这三个谓词，但这时它们的值都为假。

下例中的触发器 LogChanges 使用这三个谓词来记录表 OTHER.READER 发生的所有变化。除了记录这些信息外，它还记录对表进行变更的用户名。该触发器的记录存放在表 OTHER.READERAUDIT 中。

触发器 LogChanges 的创建语句如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER LogChanges
AFTER INSERT OR DELETE OR UPDATE ON OTHER.READER
```

```
FOR EACH ROW
DECLARE
v_ChangeType CHAR(1);
BEGIN
/* ‘I’表示 INSERT 操作，’D’表示 DELETE 操作，’U’表示 UPDATE 操作 */
IF INSERTING THEN
v_ChangeType := 'I';
ELSIF UPDATING THEN
v_ChangeType := 'U';
ELSE
v_ChangeType := 'D';
END IF;
/* 记录对 Reader 做的所有修改到表 ReaderAudit 中，包括修改人和修改时间 */
INSERT INTO OTHER.READERAUDIT
VALUES
(v_ChangeType, USER, SYSDATE,
:old.reader_id, :old.name, :old.age, :old.gender, :old.major,
:new.reader_id, :new.name, :new.age, :new.gender, :new.major);
END;

SET SCHEMA SYSDBA;
```

14.1.2 事件触发器

14.1.2.1 事件触发器语法

用户可使用触发器定义语句(CREATE TRIGGER)在数据库全局对象上创建触发器。下面是触发器定义语句的语法：

语法格式

```
CREATE [OR REPLACE] TRIGGER [<模式名>.]<触发器名> [WITH  ENCRYPTION]
BEFORE| AFTER <触发事件子句> ON <触发对象名>[WHEN <条件表达式>]<触发器体>
<触发事件子句>:=<DDL 事件子句>| <系统事件子句>
<DDL 事件子句>:=<DDL 事件>{OR <DDL 事件>}
<DDL 事件>:=DDL|<<CREATE>|<ALTER>|<DROP>|<GRANT>|<REVOKE>|<TRUNCATE>
|<COMMENT>>
<系统事件子句>:=<系统事件>{OR <系统事件>}
<系统事件>:= <LOGIN> | <LOGOUT> | <SERERR>|<BACKUP DATABASE>
|<RESTORE DATABASE>|<AUDIT>|<NOAUDIT>|<TIMER>|<STARTUP>|<SHUTDOWN>
<触发对象名>:=[<模式名>.]SCHEMA|DATABASE
```

参数

1. <模式名> 指明被创建的触发器的所在的模式名称或触发事件发生的对象所在的模式名，缺省为当前模式；
2. <触发器名> 指明被创建的触发器的名称；
3. BEFORE 指明触发器在执行触发语句之前激发；

4. AFTER 指明触发器在执行触发语句之后激发；

5.<DDL 触发事件子句> 指明激发触发器的 DDL 事件，可以是 DDL 或 CREATE、ALTER、DROP、GRANT、REVOKE、TRUNCATE、COMMENT 等；

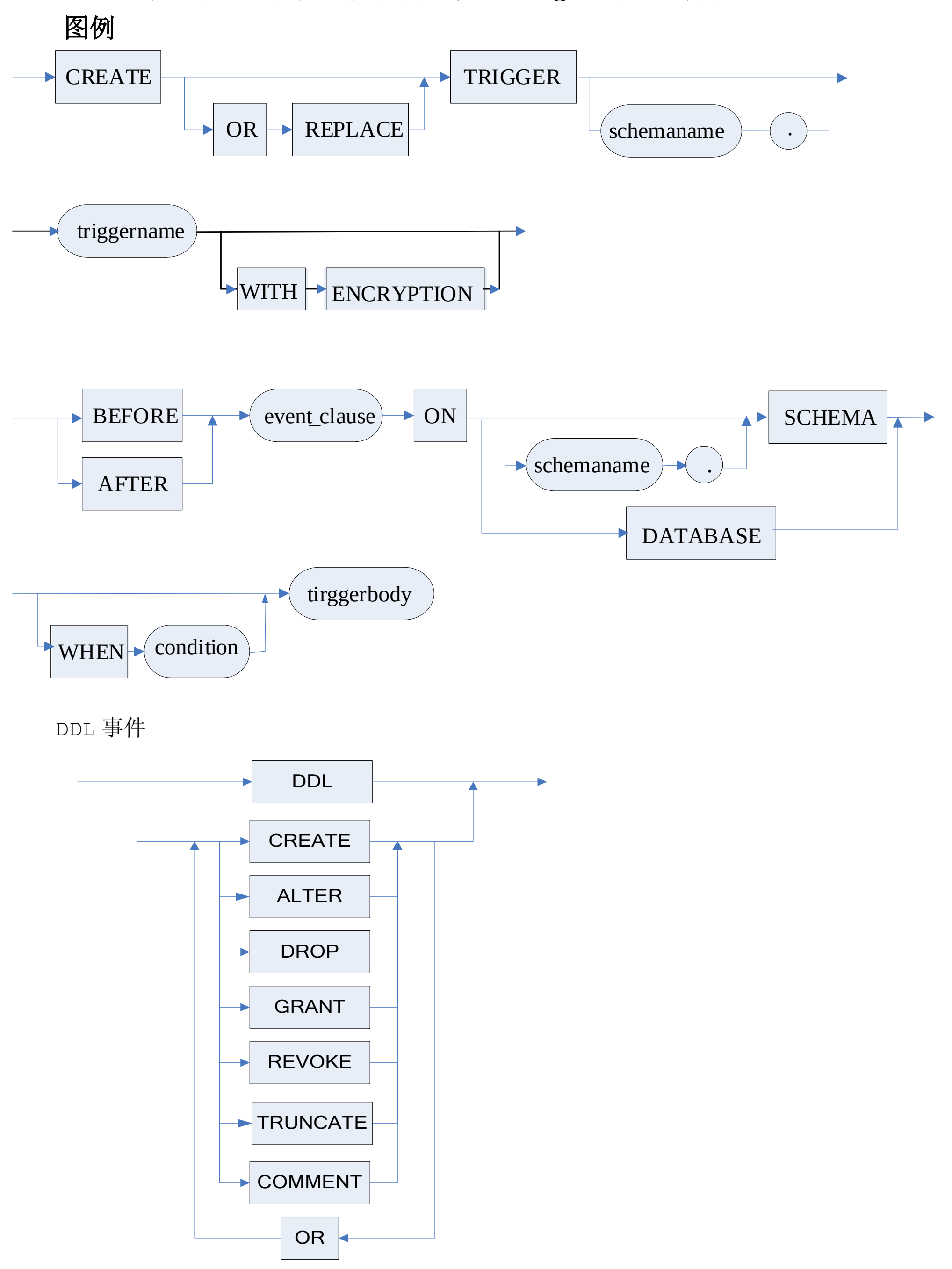
6.<系统事件子句> LOGIN/LOGON、LOGOUT/LOGOFF、SERERR、BACKUP DATABASE、RESTORE DATABASE、AUDIT、NOAUDIT、TIMER、STARTUP、SHUTDOWN；

7. WITH ENCRYPTION 选项，指定是否对触发器定义进行加密；

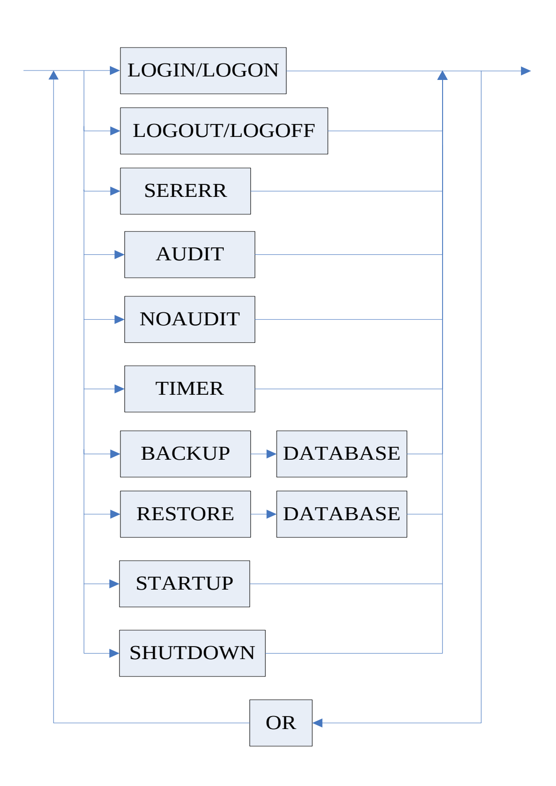
8. WHEN 子句 只允许为元组级触发器指定 WHEN 子句，它包含一个布尔表达式，当表达式的值为 TRUE 时，执行触发器；否则，跳过该触发器；

9. <触发器体> 触发器被触发时执行的 SQL 过程语句块。

图例



系统事件



使用说明

1. <触发器名>是触发器的名称，它不能与模式内的其他模式级对象同名；

2. 可以使用 OR REPLACE 选项来替换一个触发器，但是要注意被替换的触发器的触发对象名不能改变。如果要在模式中不同的对象上重新创建一个同名的触发器，则必须先删除该触发器，然后再创建；

3. <触发事件子句>说明激发触发器的事件，DDL 事件以及系统事件。DDL 事件包括数据库和模式上的 DDL 操作；系统事件包括数据库上的除 DDL 操作以外系统事件；以上事件可以有多个，用 OR 列出。触发事件按照兼容性可以分为以下几个集合：

{CREATE, ALTER, DROP, TRUNCATE,COMMENT }、{ GRANT, REVOKE }、{ LOGIN/LOGON, LOGOUT/LOGOFF }、{ SERERR }、{ BACKUP DATABASE, RESTORE DATABASE }、{AUDIT, NOAUDIT}、{TIMER}、{ STARTUP, SHUTDOWN }。

只有同一个集合中，不同名的事件，才能在创建语句中并列出现。

DDL 事件中，DDL 关键字的作用相当于 CREATE OR DROP OR ALTER OR TRUNCATE OR COMMENT。

4. <触发对象名>是触发事件发生的对象，DATABASE 和<模式名>只对 DDL 事件有效，<模式名>可以缺省；

5. 在一个数据库或模式上创建的事件触发器个数没有限制，可以有以下类型：CREATE、ALTER、DROP、GRANT、REVOKE、TRUNCATE、COMMENT、LOGIN/LOGON、LOGOUT/LOGOFF、SERERR、BACKUP DATABASE、STARTUP、SHUTDOWN，且仅表示指该类操作，不涉及到具体数据库对象如 CREATE/ALTER/DROP TABLE，只要能引起任何数据字典表中的数据对象变化，都可以激发相应触发器，触发时间分为 BEFORE 和 AFTER；所有 DDL 事件触发器都可以设置 BEFORE 或 AFTER 的触发时机，但系统事件中 LOGOUT 和 SHUTDOWN 仅能设置为 BEFORE，而其它则只能设置为 AFTER。模式级触发器不能是 LOGIN/LOGON、LOGOUT/LOGOFF、SERERR、BACKUP DATABASE、RESTORE DATABASE、STARTUP 和 SHUTDOWN 事件触发器。

6. 通过系统存储过程 SP_ENABLE_EVT_TRIGGER 和 SP_ENABLE_ALL_EVT_TRIGGER 可以禁用/启用指定的事件触发器或所有的事件触发器。

7. 事件操作说明如下：

对于事件触发器，所有的事件信息都通过伪变量 :EVENTINFO 来取得。

下面对每种事件可以获得的信息进行详细说明：

1) CREATE：添加新的数据库对象(包括用户、基表、视图等)到数据字典时触发；

```
对象类型描述：:eventinfo.objecttype
指明事件对象的类型，类型为 VARCHAR(128)，对于不同的类型其值如下：
用户： ‘USER’
表：   ‘TABLE’
视图： ‘VIEW’
索引： ‘INDEX’
过程： ‘PROCEDURE’
函数： ‘FUNCTION’
角色： ‘ROLE’
模式： ‘SCHEMA’
序列： ‘SEQUENCE’
触发器：‘TRIGGER’
同义词：‘SYNONYM’
包：‘PACKAGE’
类：‘CLASS’
类型：‘TYPE’
包体：‘PACKAGEBODY’
类体：‘CLASSBODY’
类型体：‘TYPEBODY’
表空间：'TABLESPACE'
HTS 表空间：'HUGETABLESPACE'
域：'DOMAIN'
目录：'DIRECTORY'
外部链接：'LINK'
对象名称：    :eventinfo.objectname
    指明事件对象的名称，类型为 VARCHAR(128)
所属模式：    :eventinfo.schemaname
    指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空
所属数据库：  :eventinfo.databasename
    指明事件对象所属的数据库名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空
操作类型： :eventinfo.optype
    指明事件的操作类型，类型为 VARCHAR(20)，其值为“CREATE”
操作用户名： :eventinfo.opuser
    指明事件操作者的用户名，类型为 VARCHAR(128)
事件发生时间：:eventinfo.optime
    指明事件发生的时间，类型为 DATETIME
```

2) ALTER：只要ALTER修改了数据字典中的数据对象(包括用户、基表、视图等)，就激活触发器；

```
对象类型描述： :eventinfo.objecttype
指明事件对象的类型，类型为 CHAR(1)，对于不同的类型其值如下：
用户： 'USER'
表：   'TABLE'
视图： 'VIEW'
索引： 'INDEX'
过程： 'PROCEDURE '
函数： 'FUNCTION'
序列： 'SEQUENCE'
触发器：'TRIGGER'
表空间：'TABLESPACE'
HTS 表空间：'HUGETABLESPACE'
对象名称：     :eventinfo.objectname
    指明事件对象的名称，类型为 VARCHAR(128)
所属模式：     :eventinfo.schemaname
    指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空
所属数据库：   :eventinfo.databasename
    指明事件对象所属的数据库名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空
操作类型： :eventinfo.optype
    指明事件的操作类型，类型为 VARCHAR(20)，其值为"ALTER"
操作用户名：   :eventinfo.opuser
    指明事件操作者的用户名，类型为 VARCHAR(128)
事件发生时间： :eventinfo.optime
    指明事件发生的时间，类型为 DATETIME
```

3) DROP：从数据字典删除数据库对象(包括用户、登录、基表、视图等)时触发；

```
对象类型描述： :eventinfo.objecttype
指明事件对象的类型，类型为 VARCHAR(128)，对于不同的类型其值如下：
用户： 'USER'
表：   'TABLE'
视图： 'VIEW'
索引： 'INDEX'
过程： 'PROCEDURE'
函数： 'FUNCTION'
角色： 'ROLE'
模式： 'SCHEMA'
序列： 'SEQUENCE'
触发器：'TRIGGER'
同义词：'SYNONYM'
包：'PACKAGE'
类：'CLASS'
类型：'TYPE'
表空间：'TABLESPACE'
HTS 表空间：'HUGETABLESPACE'
域：'DOMAIN'
```

```
目录：'DIRECTORY'
外部链接：'LINK'
对象名称：      :eventinfo.objectname
    指明事件对象的名称，类型为 VARCHAR(128)
所属模式：      :eventinfo.schemaname
    指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空
所属数据库：    :eventinfo.databasename
    指明事件对象所属的数据库名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空
操作类型：  :eventinfo.optype
    指明事件的操作类型，类型为 VARCHAR(20)，其值为"DROP"
操作用户名：    :eventinfo.opuser
    指明事件操作者的用户名，类型为 VARCHAR(128)
事件发生时间：  :eventinfo.optime
    指明事件发生的时间，类型为 DATETIME
```

4） GRANT：执行GRANT命令时触发；

```
权限类型描述：  :eventinfo.granttype，对于不同的类型其值如下：
     对象权限：'OBJECT_PRIV'
     系统权限：'SYSTEM_PRIV'
     角色权限：'ROLE_PRIV'
     指明授予权限的类型，类型为 varchar(256)
授予权限对象的用户名：  :eventinfo.grantee
    指明授予权限的对象用户，类型为 varchar(256)
    对象名称：:eventinfo.objectname
    对象权限有效，指明事件对象的名称，类型为 VARCHAR(128)
    所属模式：:eventinfo.schemaname
    对象权限有效，指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有
可能为空
所属数据库：    :eventinfo.databasename
    指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空
操作用户名：    :eventinfo.opuser
    指明事件操作者的用户名，类型为 VARCHAR(256)
事件发生时间：:eventinfo.optime
     指明事件发生的时间，类型为 DATETIME
对象名称：     :eventinfo.objectname
    指明事件对象的名称，类型为 VARCHAR(128)
所属模式：     :eventinfo.schemaname
    指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空
```

5） REVOKE：执行REVOKE命令时触发；

```
权限类型描述：  :eventinfo.granttype，对于不同的类型其值如下：
     对象权限：'OBJECT_PRIV'
     系统权限：'SYSTEM_PRIV'
     角色权限：'ROLE_PRIV'
     指明回收权限的类型，类型为 varchar(256)
授予权限对象的用户名：  :eventinfo.grantee
```

指明回收权限的对象用户，类型为 varchar(256)

对象名称：:eventinfo.objectname

对象权限有效，指明事件对象的名称，类型为 VARCHAR(128)

所属模式：:eventinfo.schemaname

对象权限有效，指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空

操作用户名： :eventinfo.opuser

指明事件操作者的用户名，类型为 VARCHAR(256)

所属数据库： :eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

对象名称： :eventinfo.objectname

指明事件对象的名称，类型为 VARCHAR(128)

所属模式： :eventinfo.schemaname

指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空

6) TRUNCATE：执行TRUNCATE命令时触发；

对象名称： :eventinfo.objectname，对于不同的类型其值如下：

表： 'TABLE'

指明事件对象的名称，类型为 VARCHAR(256)

所属模式： :eventinfo.schemaname

指明事件对象所属的模式名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空

所属数据库： :eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空

操作类型： :eventinfo.optype

指明事件的操作类型，类型为 VARCHAR(20)，其值为"TRUNCATE"

操作用户名： :eventinfo.opuser

指明事件操作者的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

7) LOGIN/LOGON：登录时触发；

登录名： :eventinfo.loginname

指明登录时的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

8) LOGOUT/LOGOFF：退出时触发；

登录名： :eventinfo.loginname

指明退出时的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

9) BACKUP DATABASE:备份数据库时触发；

备份的数据库：:eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

备份名： :eventinfo.backuname

指明的备份名，类型为 VARCHAR(256)

操作用户名： :eventinfo.opuser

指明事件操作者的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

10) RESTORE DATABASE:还原数据库时触发；

还原的数据库： :eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

还原的备份名：:eventinfo.backuname

指明的备份名，类型为 VARCHAR(256)

操作用户名： :eventinfo.opuser

指明事件操作者的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

11) SERERR：只要服务器记录了错误消息就触发；

错误号： :eventinfo. ERRCODE

指明错误的错误号，类型为 INT

错误信息： :eventinfo.errmsg

指明错误的错误信息，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

12) COMMENT ON DATABASE/SCHEMA：执行COMMENT命令时触发；

操作类型： :eventinfo.objecttype

指明事件对象类型，类型为 VARCHAR(20)

对象名称： :eventinfo.objectname

指明事件对象的名称，类型为 VARCHAR(128)

所属模式： :eventinfo.schemaname

指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空

所属数据库： :eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空

操作类型： :eventinfo.optype

指明事件的操作类型，类型为 VARCHAR(20)，其值为"COMMENT"

操作用户名： :eventinfo.opuser

指明事件操作者的用户名，类型为 VARCHAR(128)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

13) AUDIT：进行审计时触发（用于收集，处理审计信息）；

14) NOAUDIT：不审计时触发；

15）TIMER：定时触发。见下文时间触发器；

16）STARTUP：服务器启动后触发，只能 AFTER STARTUP。

SHUTDOWN：服务器关闭前触发，只能BEFORE SHUTDOWN。SHUTDOWN触发，不要执行花费时间多于5秒的操作。

7. <触发器体>是触发器的执行代码，是一个 DMSQL 程序语句块，语句块与存储模块定义语句中<模块体>的语法基本相同。有关详细语法，可参考第 10 章的相关部分。<引用子句>

用来引用正处于修改状态下表中行的数据。如果指定了<触发条件>子句，则首先对该条件表达式求值，<触发器体>只有在该条件为真值时才运行；

8．创建模式触发器时，触发对象名直接用 SCHEMA；

9．创建的触发器可以分为以下几类：

1） 在自己拥有的模式中创建自己模式的对象上的触发器或创建自己模式上的触发器；

2） 在任意模式中创建任意模式的对象上的触发器或创建其他用户模式上(. SCHEMA)的触发器，即支持跨模式的触发器，表现为<触发器名>和<触发对象名>的<模式名>不同；

3） 创建数据库上(DATABASE)的触发器。

10．触发器的创建者必须拥有 CREATE TRIGGER 数据库权限并具有触发器定义中引用对象的权限；

11．DDL 触发事件的用户必须拥有对模式或数据库上相应对象的 DDL 权限；系统触发事件的用户必须有 DBA 权限；

12．如果触发器执行 SQL 语句或调用过程或函数，那么触发器的拥有者必须拥有执行这些操作所必需的权限。这些权限必须直接授予触发器拥有者，而不是通过角色授予，这与存储模块或函数的限制一致；

13．如果触发器同时也是触发事件对象，则该触发器不会被激发，例如：当删除触发器本身被删除时不会触发 DROP 触发器。

权限

用户必须是基表的拥有者，或者具有 DBA 权限。

需要强调的是，由于触发器是激发它们的语句的一个组成部分，为保证语句的原子性，在<触发器体>以及<触发器体>调用的存储模块中不允许使用可能导致事务提交或回滚的 SQL 语句，如：COMMIT、ROLLBACK。

具体地说，在触发器中允许的 SQL 语句有： SELECT、INSERT、DELETE、UPDATE、DECLARE CURSOR、OPEN、FETCH、CLOSE 语句等。

每张基表上的可创建的触发器的个数没有限制，但是触发器的个数越多，处理 DML 语句所需的时间就越长，这是显而易见的。注意，不存在触发器的执行权限，因为用户不能主动“调用”某个触发器，是否激发一个触发器是由系统来决定的。

事件触发器（DDL 触发事件）使用示例如下：

```
SQL>CREATE TABLE T01_TRI_10000(OBJECTTYPE VARCHAR(500),OBJECTNAME VARCHAR(500), SCHEMANAME VARCHAR(500),DATABASENAME VARCHAR(500),OPUSER VARCHAR(500), OPTIME VARCHAR(500));

SQL>CREATE TABLE T02_TRI_10000 (C1 INT,C2 VARCHAR(10));
SQL>INSERT INTO T02_TRI_10000 VALUES (1,'ABCD');
SQL>CREATE TRIGGER TRI01_TRI_10000 BEFORE CREATE ON DATABASE BEGIN INSERT INTO T01_TRI_10000 VALUES(:EVENTINFO.OBJECTTYPE,:EVENTINFO.OBJECTNAME,:EVENTINFO.SCHEMANAME,:EVENTINFO.DATABASENAME,:EVENTINFO.OPUSER, :EVENTINFO.OPTIME); END;

SQL>CREATE USER L01_TRI_10000 IDENTIFIED BY L01_TRI_10000;
SQL>CREATE TABLE T03_TRI_10000(C1 INT);
SQL>CREATE VIEW V01_TRI_10000 AS SELECT * FROM T01_TRI_10000;
SQL>CREATE INDEX I01_TRI_10000 ON T01_TRI_10000(OBJECTTYPE);
```

```
SQL>CREATE OR REPLACE PROCEDURE P01_TRI_10000 AS BEGIN SELECT * FROM T02_TRI_10000; END;

SQL>CREATE FUNCTION F01_TRI_10000 RETURN VARCHAR(30) AS A1 VARCHAR(30); BEGIN SELECT C2 INTO A1 FROM T02_TRI_10000 WHERE C1=1; PRINT A1; RETURN A1; END;
SQL>CREATE ROLE R01_TRI_10000;
SQL>CREATE SEQUENCE S01_TRI_10000 INCREMENT BY 10;
SQL>CREATE TRIGGER TRI02_TRI_10000 AFTER CREATE ON DATABASE BEGIN PRINT 'SUCCESS';END;

SQL>SELECT OBJECTTYPE, OBJECTNAME, SCHEMANAME, DATABASENAME, OPUSER FROM T01_TRI_10000;
```

查询结果如下：

```
行号     OBJECTTYPE     OBJECTNAME SCHEMANAME DATABASENAME OPUSER
----------  --------------------    ----------------------
1           USER      L01_TRI_10000      NULL      DAMENG    SYSDBA
2           TABLE      T03_TRI_10000     SYSDBA      DAMENG    SYSDBA
3           VIEW      V01_TRI_10000     SYSDBA      DAMENG    SYSDBA
4           INDEX       I01_TRI_10000     SYSDBA      DAMENG    SYSDBA
5           PROCEDURE      P01_TRI_10000     SYSDBA      DAMENG    SYSDBA
6           FUNCTION      F01_TRI_10000     SYSDBA      DAMENG    SYSDBA
7           ROLE      R01_TRI_10000      NULL      DAMENG    SYSDBA
8           SEQUENCE      S01_TRI_10000     SYSDBA      DAMENG    SYSDBA
9           TRIGGER     TRI02_TRI_10000     SYSDBA      DAMENG    SYSDBA
```

事件触发器（系统触发事件）使用示例如下：

```
create or replace trigger test_trigger after LOGIN on database begin print'SUCCESS'; end;
----只要一登录，服务器就会打印出 SUCCESS
```

14.1.2.2 事件属性函数用法

当事件触发器被触发时，可以通过这些事件属性函数获取当前事件的属性。

针对用户设置的数据库事件(DDL 语句执行)，获取事件触发时的相关属性。事件属性函数如下：

1、DM_DICT_OBJ_NAME，无参数，返回事件对象名；

2、DM_DICT_OBJ_TYPE，无参数，返回事件对象类型；

3、DM_DICT_OBJ_OWNER，无参数，返回事件对象所在模式；

4、DM_SQL_TXT，有 1 个输出参数，参数类型为 DM_NAME_LIST_T，返回值为 DDL 语句占用的嵌套表单元个数。DM_SQL_TXT 帮助用户获取事件被触发时正在执行的 DDL 语句，用于存储获取到的 DDL 语句。DM_NAME_LIST_T 为元素类型为 varchar(64)的嵌套表。因此如果 DDL 语句过长会导致分片存储，用户在获取 DDL 语句的时候，尤其要注意根据返回值来循环读取嵌套表以获取完整的语句。

使用说明

1、系统内部 DDL 将不触发事件触发器

2、MPP 从节点的 DDL 将不触发事件触发器

下面用一个具体的例子来说明事件属性函数如何使用。

```
CREATE TABLE T_EAF(
    N       INT,
    SQLTEXT VARCHAR,
    OBJECTNAME VARCHAR(128),
    OBJECTTYPE VARCHAR(128),
    OBJECTOWNER   VARCHAR(128)
);
CREATE OR REPLACE TRIGGER TRIG_EAF_01 BEFORE DDL ON DATABASE
   DECLARE
      N       NUMBER;
      STR_STMT VARCHAR;
      SQL_TEXT DM_NAME_LIST_T;
   BEGIN
      N := DM_SQL_TXT(SQL_TEXT);  --N 为占用嵌套表单元个数
       FOR I IN 1..N
      LOOP
        STR_STMT := STR_STMT || SQL_TEXT(I);  -- STR_STMT 为获取的 DDL 语句
      END LOOP;
     INSERT INTO T_EAF VALUES(N,STR_STMT,DM_DICT_OBJ_NAME, DM_DICT_OBJ_TYPE,
DM_DICT_OBJ_OWNER);
END;
/
```

执行建模式建表语句。

```
create schema systest;
create table T_systest(c1 int);
```

然后，可以在 T_EAF 中查询到相关的建模式、建表语句。

```
SELECT * FROM T_EAF;
```

查询结果：

```
N   SQLTEXT                         OBJECTNAME OBJECTTYPE  OBJECTOWNER
1   create schema systest;          SYSTEST    SCHEMA      SYSTEST
1   create table T_systest(c1 int); T_SYSTEST  TABLE        SYSDBA
```

14.1.3 时间触发器

时间触发器属于一种特殊的事件触发器，它使得用户可以定义一些有规律性执行的、定点执行的任务，比如在晚上服务器负荷轻的时候通过时间触发器做一些更新统计信息的操作、自动备份操作等等，因此时间触发器是非常有用的。

语法格式

```
    CREATE [OR REPLACE] TRIGGER [<模式名>.]<触发器名>[WITH  ENCRYPTION]
    AFTER TIMER ON DATABASE
    <{FOR ONCE AT DATETIME [时间表达式]}|{{<month_rate>|<week_rate>|<day_rate>}
{once_in_day|times_in_day}{during_date}}>
```

```
[WHEN <条件表达式>]
<触发器体>
<month_rate>:= {FOR EACH <整型变量> MONTH {day_in_month}}| {FOR EACH <整型变量> MONTH { day_in_month_week}}
<day_in_month>:= DAY <整型变量>
<day_in_month_week>:= {DAY <整型变量> OF WEEK<整型变量>}|{DAY <整型变量> OF WEEK LAST}
<week_rate>: =FOR EACH <整型变量> WEEK {day_of_week_list}
< day_of_week_list >:= {<整型变量>}|{, <整型变量>}
<day_rate>: =FOR EACH <整型变量> DAY
< once_in_day >:= AT TIME <时间表达式>
< times_in_day >: ={ duaring_time } FOR EACH <整型变量> MINUTE
<duaring_time>: ={NULL}|{FROM TIME <时间表达式>}|{FROM TIME <时间表达式> TO TIME <时间表达式>}
<duaring_date>: ={NULL}|{FROM DATETIME <日期时间表达式>}|{FROM DATETIME <日期时间表达式> TO DATETIME <日期时间表达式>}
```

参数

1. <模式名>　指明被创建的触发器的所在的模式名称或触发事件发生的对象所在的模式名，缺省为当前模式；
2. <触发器名> 指明被创建的触发器的名称；
3. WHEN 子句　包含一个布尔表达式，当表达式的值为 TRUE 时，执行触发器；否则，跳过该触发器；
4. <触发器体>　触发器被触发时执行的 SQL 过程语句块。

时间触发器的最低时间频率精确到分钟级，定义很灵活，完全可以实现数据库中的代理功能，只要通过定义一个相应的时间触发器即可。在触发器体中定义要做的工作，可以定义操作的包括执行一段 SQL 语句、执行数据库备份、执行重组 B 树、执行更新统计信息、执行数据迁移（DTS）。

下面的简单例子在屏幕上每隔一分钟输出一行"HELLO WORLD"。

```
CREATE OR REPLACE TRIGGER timer2
AFTER TIMER on database
for each 1 day for each 1 minute
BEGIN
    print 'HELLO WORLD';
END;
/
```

关闭时间触发器和普通触发器是一样的，这里不再叙述。

14.2 触发器替换

在定义触发器的语法中，"OR REPLACE"选项用于替换一个已存在的同名触发器。当触发器替换是以下情况之一时，DM 会报错"替换触发器属性不一致"。

1） 表触发器和事件触发器之间的替换；
2） 表触发器所基于的表或视图发生变化时；

3） 事件触发器的触发对象名（SCHEMA 或 DATABASE）发生变化时；
4） 事件触发器的可触发的模式发生变化时；
5） 事件触发器对应激发触发器的事件类型发生变化时，事件类型分为以下几类：
DDL：CREATE、ALTER、DROP、GRANT、REVOKE、TRUNCATE 等
AUDIT：AUDIT、NOAUDIT
PRIV：GRANT、REVOKE
LOGIN： LOGIN/LOGON、LOGOUT/LOGOFF
SERVER：SERERR
BACK：BACKUP DATABASE、RESTORE DATABASE
TIMER：TIMER
STARTUP：STARTUP、SHUTDOWN

14.3 设计触发器的原则

在应用中使用触发器功能时，应遵循以下设计原则，以确保程序的正确和高效：

1. 如果希望保证一个操作能引起一系列相关动作的执行，请使用触发器；
2. 不要用触发器来重复实现 DM 中已有的功能。例如，如果用约束机制能完成希望的完整性检查，就不要使用触发器；
3. 避免递归触发。所谓递归触发，就是触发器体内的语句又会激发该触发器，导致语句的执行无法终止。例如，在表 T1 上创建 BEFORE UPDATE 触发器，而该触发器中又有对表 T1 的 UPDATE 语句；
4. 合理地控制触发器的大小和数目。要知道，一旦触发器被创建，任何用户在任何时间执行的相应操作都会导致触发器的执行，这将是一笔不小的开销。

14.4 触发器的删除

当用户需要从数据库中删除一个触发器时，可以使用触发器删除语句。其语法如下：

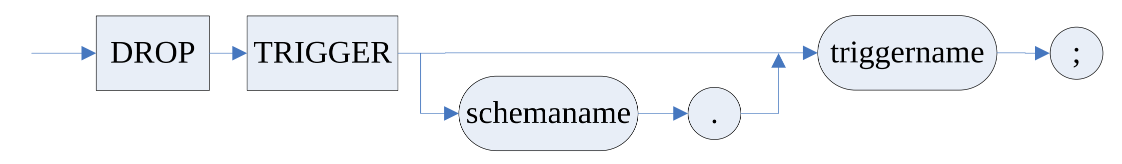
语法格式

```
DROP TRIGGER [<模式名>.]<触发器名>;
```

参数

1. <模式名> 指明被删除触发器所属的模式；
2. <触发器名> 指明被删除的触发器的名字。

图例



使用说明

1. 当触发器的触发表被删除时，表上的触发器将被自动地删除；
2. 除了 DBA 用户外，其他用户必须是该触发器所属基表的拥有者才能删除触发器。

权限

执行该操作的用户必须是该触发器所属基表的拥有者，或者具有 DBA 权限。

举例说明

例 1 删除触发器 TRG1。

```
DROP TRIGGER TRG1;
```

例 2 删除模式 SYSDBA 下的触发器 TRG2。

```
DROP TRIGGER SYSDBA.TRG2;
```

14.5 禁止和允许触发器

每个触发器创建成功后都自动处于允许状态(ENABLE)，只要基表被修改，触发器就会被激发。但是在某些情况下，例如：

1. 触发器体内引用的某个对象暂时不可用；
2. 载入大量数据时，希望屏蔽触发器以提高执行速度；
3. 重新载入数据。

用户可能希望触发器暂时不被触发，但是又不想删除这个触发器。这时，可将其设置为禁止状态(DISABLE)。

当触发器处于允许状态时，只要执行相应的 DML 语句，且触发条件计算为真，触发器体的代码就会被执行；当触发器处于禁止状态时，则在任何情况下触发器都不会被激发。根据不同的应用需要，用户可以使用触发器修改语句将触发器的状态设置为允许或禁止状态。其语法如下：

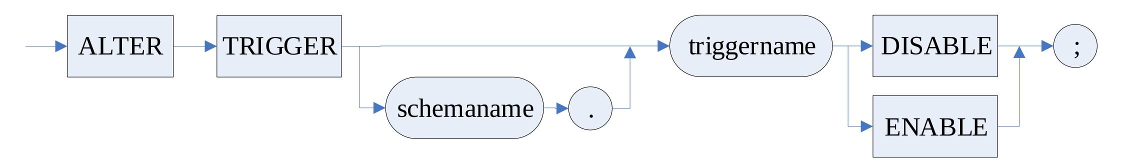
语法格式

```
ALTER TRIGGER [<模式名>.]<触发器名> <DISABLE | ENABLE>;
```

参数

1. <模式名> 指明被修改的触发器所属的模式；
2. <触发器名> 指明被修改的触发器的名字；
3. DISABLE 指明将触发器设置为禁止状态。当触发器处于禁止状态时，在任何情况下触发器都不会被激发；
4. ENABLE 指明将触发器设置为允许状态。当触发器处于允许状态时，只要执行相应的 DML 语句，且触发条件计算为真，触发器就会被激发。

图例



14.6 触发器的重编

重新对触发器进行编译，如果重新编译失败，则将触发器置为禁止状态。

重编功能主要用于检验触发器的正确性。

语法格式

```
ALTER TRIGGER [<模式名>.]<触发器名> COMPILE [DEBUG];
```

参数

1. <模式名> 指明被重编译的触发器所属的模式；
2. <触发器名> 指明被重编译的触发器的名字；
3. [DEBUG] 可忽略。

图例

ALTER → TRIGGER → [schemaname .] → triggername → COMPILE → [DEBUG] → ;

权限

执行该操作的用户必须是触发器的创建者，或者具有 DBA 权限。

举例说明

例 重新编译触发器

```
ALTER TRIGGER OTHER.TRG_AI_ACCOUNT COMPILE;
```

14.7 触发器应用举例

正如我们在本章所介绍的，触发器是 DM 系统提供的重要机制。我们可以使用该机制来加强比正常的审计机制、完整性约束机制、安全机制等所能提供的功能更复杂的事务规则。为帮助用户更好地使用该机制，我们提供了一些触发器应用的例子供用户参考。

14.7.1 使用触发器实现审计功能

尽管 DM 系统本身已经提供了审计机制，但是在许多情况下我们还是可以利用触发器完成条件更加复杂的审计。与内置的审计机制相比，采用触发器实现的审计有如下优点：

1．使用触发器可针对更复杂的条件进行审计；

2．使用触发器不仅可以记录操作语句本身的信息，还可以记录被该语句修改的数据的具体值；

3．内置的审计机制将所有审计信息集中存放，而触发器实现的审计可针对不同的操作对象分别存放审计信息，便于分析。

虽然如此，触发器并不能取代内置的审计机制。因为内置审计机制的某些功能触发器是无法做到的。例如：

1. 内置审计机制可审计的类型更多。触发器只能审计表上的 DML 操作，而内置审计机制可以针对各种操作、对象和用户进行审计；
2. 触发器只能审计成功的操作，而内置审计机制能审计失败的操作；
3. 内置审计机制使用起来更简单，并且其正确性更有保障。

用于审计的触发器通常都是 AFTER 类型。关于审计的实例，请参考《DM8 安全管理》。

14.7.2 使用触发器维护数据完整性

触发器与完整性约束机制都可以用于维护数据的完整性，但是二者之间存在着显著的区别。一般情况下，如果使用完整性约束机制可以完成约束检查，我们不建议用户使用触发器。这是因为：

1. 完整性约束机制能保证表上所有数据符合约束，即使是约束创建前存在的数据也必须如此；而触发器只保证其创建后的数据满足约束，但之前存在数据的完整性则得不到保证；

2. 完整性约束机制使用起来更简单，并且其正确性更有保障。

触发器通常用来实现完整性约束机制无法完成的约束检查和维护，例如：

1. 引用完整性维护

删除被引用表中的数据时，级联删除引用表中引用该数据的记录；更新被引用表中的数据时，更新引用表中引用该数据的记录的相应字段。下例中，表 OTHER.DEPTTAB 为被引用表，其主关键字为 Deptno；表 OTHER.EMPTAB 为引用表。其结构如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Dept_del_upd_cascade
AFTER DELETE OR UPDATE ON OTHER.DEPTTAB FOR EACH ROW
BEGIN
   IF DELETING THEN
      DELETE FROM OTHER.EMPTAB
         WHERE Deptno = :old.Deptno;
   ELSE
      UPDATE OTHER.EMPTAB SET Deptno = :new.Deptno
         WHERE Deptno = :old.Deptno;
   END IF;
END;

SET SCHEMA SYSDBA;
```

2. CHECK 规则检查

增加新员工或者调整员工工资时，保证其工资不超过规定的范围，并且涨幅不超过 25%。该例中，表 OTHER.EMPTAB 记录员工信息；表 OTHER.SALGRADE 记录各个工种的工资范围，其结构如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Salary_check
BEFORE INSERT OR UPDATE ON OTHER.EMPTAB
FOR EACH ROW
DECLARE
   Minsal        FLOAT;
   Maxsal        FLOAT;
   Salary_out_of_range  EXCEPTION FOR -20002;
BEGIN
```

```
/* 取该员工所属工种的工资范围 */
SELECT Losal, Hisal INTO Minsal, Maxsal FROM OTHER.SALGRADE
WHERE Job_classification = :new.Job;
/* 如果工资超出工资范围，报告异常 */
IF (:new.Sal < Minsal OR :new.Sal > Maxsal) THEN
     RAISE Salary_out_of_range;
END IF;
/* 如果工资涨幅超出 25%，报告异常 */
IF UPDATING AND (:new.Sal - :old.Sal) / :old.Sal > 0.25 THEN
     RAISE Salary_out_of_range;
END IF;
END;

SET SCHEMA SYSDBA;
```

14.7.3 使用触发器保障数据安全性

在复杂的条件下，可以使用触发器来保障数据的安全性。同样，要注意不要用触发器来实现 DM 安全机制已提供的功能。使用触发器进行安全控制时，应使用语句级 BEFORE 类型的触发器，其优点如下：

1. 在执行触发事件之前进行安全检查，可以避免系统在触发语句不能通过安全检查的情况下做不必要的工作；

2. 使用语句级触发器，安全检查只需要对每个触发语句进行一次，而不必对语句影响的每一行都执行一次。

下面这个例子显示如何用触发器禁止在非工作时间内修改表 OTHER.EMPTAB 中的工资(Sal)栏。非工作时间包括周末、公司规定的节假日以及下班后的时间。为此，我们用表 OTHER.C OMPANYHOLIDAYS 来记录公司规定的节假日。其结构如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Emp_permit_changes
BEFORE INSERT OR DELETE OR UPDATE
ON OTHER.EMPTAB
DECLARE
  Dummy    INTEGER;
  Invalid_Operate_time EXCEPTION FOR -20002;
BEGIN
  /* 检查是否周末 */
  IF (DAYNAME(Sysdate) = 'Saturday' OR
    DAYNAME(Sysdate) = 'Sunday') THEN
    RAISE Invalid_Operate_time;
  END IF;
  /* 检查是否节假日 */
  SELECT COUNT(*) INTO Dummy FROM OTHER.COMPANYHOLIDAYS
    WHERE Holiday= Current_date;
```

```
    IF dummy > 0 THEN
      RAISE Invalid_Operate_time;
    END IF;
    /* 检查是否上班时间 */
    IF (EXTRACT(HOUR FROM Current_time) < 8 OR
       EXTRACT(HOUR FROM Current_time) >= 18) THEN
       RAISE Invalid_Operate_time;
    END IF;
END;

SET SCHEMA SYSDBA;
```

14.7.4 使用触发器生成字段默认值

触发器还经常用来自动生成某些字段的值，这些字段的值有时依赖于本记录中的其他字段的值，有时是为了避免用户直接对这些字段进行修改。这类触发器应该是元组级 BEFORE INSERT 或 UPDATE 触发器。因为：

1. 必须在 INSERT 或 UPDATE 操作执行之前生成字段的值；
2. 必须为每条元组自动生成一次字段的值。

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Emp_auto_value
BEFORE INSERT
ON OTHER.EMPTAB
FOR EACH ROW
BEGIN
  :new.Sal = 999.99;
END;

SET SCHEMA SYSDBA;
```

第 15 章 同义词

同义词(Synonym)让用户能够为数据库的一个模式下的对象提供别名。同义词通过掩盖一个对象真实的名字和拥有者,并且对远程分布式的数据库对象给予了位置透明特性以此来提供了一定的安全性。同时使用同义词可以简化复杂的 SQL 语句。同义词可以替换模式下的表、视图、序列、函数、存储过程等对象。

15.1 创建同义词

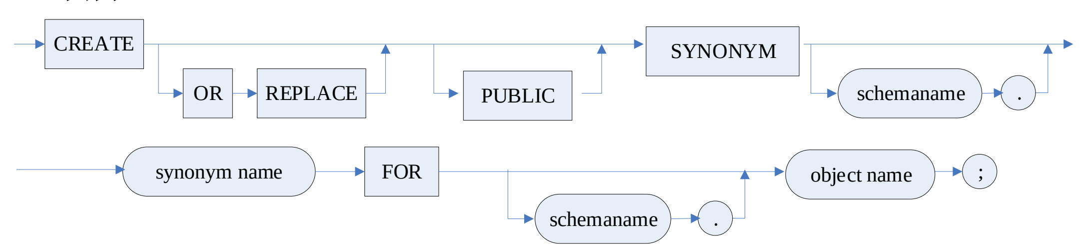
语法格式

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [<模式名>.]<同义词名> FOR [<模式名>.]<对象名>
```

参数

1. <同义词名> 指被定义的同义词的名字;
2. <对象名> 指示同义词替换的对象。

图例



语句功能

创建一个同义词。

使用说明

1. 同义词分为全局同义词(PUBLIC SYNONYM)和非全局同义词。用户在自己的模式下创建同义词,必须有 CREATE SYNONYM 权限。用户要创建全局同义词(PUBLIC SYNONYM),必须有 CREATE PUBLIC SYNONYM 权限;

2. 全局同义词创建时不能指定同义词的模式名限定词,它能够被所有用户使用,使用时不需要加任何模式限定名。非全局同义词被非同义词所属模式拥有者引用需要在前面加上模式名;公有同义词和私有同义词,可以具有相同的名字;非全局同义词不能与目标对象同名;

3. 用户使用 SQL 语句对某个对象进行操作,那么解析一个对象的顺序,首先是查看模式内是否存在该对象,然后再查看模式内的同义词(非全局同义词),最后才是全局同义词。例如,用户 OE 和 SH 在他们的模式下都有一个表叫 customer,SYSDBA 为 OE 模式下的 customer 表创建了一个全局同义词 customer_syn,SYSDBA 为 SH 模式下的 customer 表创建了一个私有同义词 customer_syn,如果用户 SH 查询:SELECT COUNT(*) FROM customer_syn,则此时返回的结果为 SH.CUSTOMER 下的行数,而如果需要访问 OE 模式下的 CUSTOMER 表,则必须在前面加模式名:SELECT COUNT(*) FROM OE.customer_syn;

4. 如果创建时候没有指定 REPLACE 语法要素，则不允许创建同名同类型同义词(不同类型则可以，这里的不同类型指的是 PUBLIC 和非 PUBLIC)；

5. 同义词创建时，并不会检查他所指代的同义词对象是否存在，用户使用该同义词时候，如果不存在指代对象或者对该指代对象不拥有权限，则会报错。

6.当 INI 参数 ENABLE_PL_SYNONYM=0 时，禁止通过全局同义词执行非系统用户创建的包或者 DMSQL 程序。

举例说明

例 1 用户 B 对 A 模式下的表 T1 创建同义词。

在 A 模式下建立表 T1。

```
CREATE TABLE "A"."T1" ("ID" INTEGER, "NAME" VARCHAR(50), PRIMARY KEY("ID"));
INSERT INTO "A"."T1" ("ID", "NAME") VALUES (1, '张三');
INSERT INTO "A"."T1" ("ID", "NAME") VALUES (2, '李四');
```

对 A 模式下的表 T1 创建同义词。

```
CREATE SYNONYM A.S1 FOR A.T1;
```

如果用户 B 想查询 T1 表的行数，可以通过如下语句来获得结果：

```
SELECT COUNT(*) FROM A.S1;
```

例 2 当 INI 参数 ENABLE_PL_SYNONYM=0 时，禁止通过全局同义词执行非系统用户创建的包或者 DMSQL 程序。

```
1. dba 登录创建新用户并赋予创建同义词权限
--SYSDBA 登录
    create user DBSEC  identified by 123456789;
    GRANT CREATE PUBLIC SYNONYM TO DBSEC;
    grant resource to DBSEC;
2. 创建同义词。通过同义词调用 DMSQL 程序。
--DBSEC 登录
    CREATE OR REPLACE FUNCTION GET_DBA2(a INT, b INT) RETURN INT AS
        s  INT;
        BEGIN
        s:=a+b;
        RETURN s;
        EXCEPTION
       WHEN OTHERS THEN NULL;
        END;
        /
    CREATE OR REPLACE PUBLIC SYNONYM SP_CLOSE_SESSION FOR GET_DBA2;--创建同义词
    call SP_CLOSE_SESSION (1,2);   --报错无法解析的成员访问表达式[SP_CLOSE_SESSION]
```

15.2 删除同义词

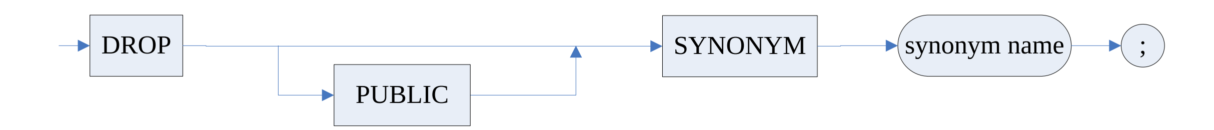
语法格式

```
DROP [PUBLIC] SYNONYM <同义词名>
```

参数

<同义词名> 指被定义的同义词的名字。

图例



语句功能

删除一个同义词。

使用说明

1. 必须是 DBA 或者拥有此同义词的用户才能删除该同义词；

2. 如果要删除公有同义词，则必须要指定 PUBLIC，因为在达梦数据库中，公有同义词和私有同义词可以同名，所以如果不指定，则删除的是当前模式下的同名同义词；

3. 删除公有同义词，需要指定 PUBLIC，而删除私有同义词，则不能指定，否则报错；如果删除当前模式下的同义词，可以不指定模式名，如果删除其它模式下的同义词，需要指定相应的模式名，否则报错。

举例说明：

例 1 删除模式 A 下的同义词 S1。

```
DROP SYNONYM A.S1;
```

例 2 删除公有同义词 S2。

```
DROP PUBLIC SYNONYM S2;
```

第 16 章 外部链接

外部链接对象（LINK）是 DM 中的一种特殊的数据库实体对象，它记录了远程数据库的连接和路径信息，用于建立与远程数据的联系。通过多台数据库主库间的相互通讯，用户可以透明地操作远程数据库的数据，使应用程序看起来只有一个大型数据库。用户远程数据库中的数据请求，都被自动转换为网络请求，并在相应结点上实现相应的操作。用户可以建立一个数据库链接，以说明一个对象在远程数据库中的访问路径。这个链接可以是公用的（数据库中所有用户使用），也可以是私有的（只能被某个用户使用）。

用户可以通过外部链接对远程数据库的表进行查询和增删改操作，以及本地调用远程的存储过程。

16.1 创建外部链接

创建一个外部链接。

语法格式

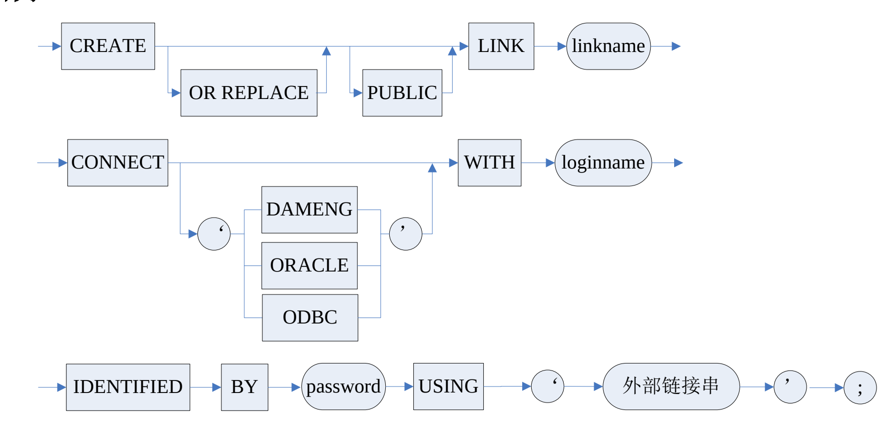
```
    CREATE [OR REPLACE] [PUBLIC] LINK <外部链接名> CONNECT ['<连接库类型>'] WITH <
登录名> IDENTIFIED BY <登录口令> USING  '<外部连接串>';
    <连接库类型> ::= DAMENG | ORACLE | ODBC
    <外部链接串>::=< DAMENG 外部链接串>| < ORACLE 外部链接串> | <ODBC 外部链接串>
    <DAMENG 外部链接串>::=[<连接类型>;]<服务器列表>
    <连接类型>::= PRIMARY FIRST |
                  STANDBY FIRST |
                  PRIMARY ONLY |
                  STANDBY ONLY
    <服务器列表>::=<服务器地址> |
                   <服务器地址>{,<服务器地址>}
    <服务器地址>::=<实例 IP 地址>/<实例端口号> |
                   <MAL IP 地址>/<MAL 端口号> |
                   <实例名>
    <ORACLE 外部链接串>::= <tsn_name> |<description>|<IP 地址>/<服务名>
    <description>::=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=
<IP 地址>)(PORT=<端口号>)))(CONNECT_DATA=(SERVICE_NAME=<服务名>)))
    <ODBC 外部链接串>::= <ODBC 数据源 DSN>
```

参数

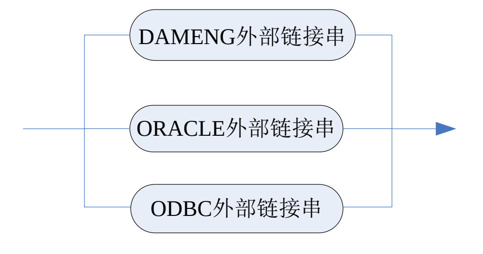
1. OR REPLACE 使用 OR REPLACE 选项的好处是，如果系统中已经有同名的数据库链接名，服务器会自动用新的代码覆盖原来的代码。如果不使用 OR REPLACE 选项，当创建的新外部链接名称与系统中已有的外部链接名称同名时，服务器会报错。
2. PUBLIC 此链接对象是否能够被创建者之外的用户引用；
3. <外部链接名> 数据库链接的名称；
4. <连接库类型> 目前只支持 DAMENG、ORACLE 或 ODBC，默认为 DAMENG；
5. <登录名> 登录用户名称；

6. <登录口令> 登录用户口令；
7. <连接类型> 当 DBLINK 连接到多机环境时，可以设定<连接类型>和指定多台<服务器地址>。<连接类型>用来指定 DBLINK 将采用何种优先级别连接到<服务器列表>中的机器，共四种连接类型：PRIMARY FIRST 为主机优先连接；STANDBY FIRST 为备机优先连接；PRIMARY ONLY 为只连主机；STANDBY ONLY 为只连备机。缺省为 PRIMARY FIRST。<连接类型>为可选项，一旦指定了<连接类型>，则必须指定<服务器地址>数大于等于 2；
8. <DAMENG 外部链接串>支持三种格式，分别对应目标节点在 dmmal.ini 中的配置项，具体如下：
   - <实例 IP 地址>/<实例端口号> 对应 mal_inst_host/mal_inst_port
   - <MAL IP 地址>/<MAL 端口号> 对应 mal_host/mal_port
   - 实例名 对应 mal_inst_name
9. <ORACLE 外部链接串>可以使用配置的网络服务名 tsn_name(网络服务名需要配置)，或者连接描述符 description（连接描述符是网络连接目标特殊格式的描述，它包括网络协议、主库 IP 地址、端口号和服务名)，或者<IP 地址>/<服务名>；
10. <ODBC 外部链接串>DSN 需要用户手动配置。

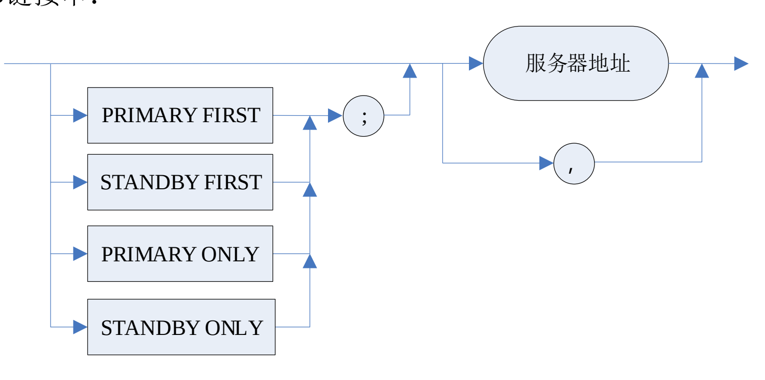
图例



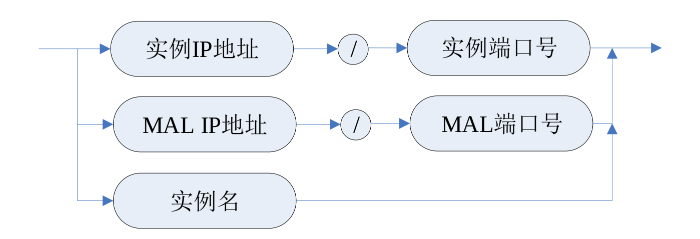
外部链接串：



DAMENG 外部链接串：



服务器地址：



语句功能

创建一个外部链接。

使用说明

1. 要创建到 DM 数据库的外部链接，必须首先配置 dmmal.ini，才能使用 LINK。DM 的连接串有两种格式：

INSTANCE_NAME:直接使用远程库的实例名（该实例名必须配置到 dmmal.ini 中）；

<IP 地址>/<端口号>:其中端口号为 DM 外部链接服务器的 dmmal.ini 配置中的 MAL_PORT 端口号。

dmmal.ini 各配置项的值、前后顺序，必须保持完全一致。dmmal.ini 的详细配置可参考《DM8 系统管理员手册》的 2.1 节，需要注意同时将 dm.ini 中的 MAL_INI 参数置为 1 以开启 MAL 系统。

2. 要创建到 ORACLE 的外部链接，可以使用配置的网络服务名<tsn_name>；如果没有配置 tsn_name，可以使用连接描述符<description>或者<IP 地址>/<服务名>作为连接串。

3. 要创建到 ORACLE 的外部链接，需要在当前机器安装 ORACLE 的 OCI 接口，且需要保证 OCI 接口与 DM 版本在 32 位/64 位保持一致。

4. 通过 LINK 对远程服务器所作的修改，由用户在本地服务器通过 commit 或 rollback 进行提交或回滚。

5. 只支持普通用户，不支持 SSL 和 Kerberos 认证。

6. DM8 不支持连接自身实例的 LINK。

7. 支持在 CREATE SCHEMA 中 CREATE LINK，但是不支持 CREATE PUBLIC LINK。

8. 只有 DBA 和具有 CREATE LINK 权限的用户可以创建外部链接。

9.当 DBLINK 链接的是多机系统且指定了多台服务器时，若所连接的服务器发生宕机等意外情况导致无法对外提供服务时，DBLINK 将根据设定的<连接类型>，在剩余的服务器中挑选合适的服务器进行连接。

举例说明

例 1 使用 DM 数据库，创建一个连接到 IP 地址为 192.168.0.31，MAL_PORT 端口号为 5369 的 MAL 站点的外部链接，登录到此站点使用的用户名为 USER01，密码为 AAA123456，实例名为：DMSERVER。

```
    CREATE PUBLIC LINK LINK1 CONNECT 'DAMENG'  WITH USER01 IDENTIFIED BY AAA123456
USING '192.168.0.31/5369';
    或
    CREATE PUBLIC LINK LINK1 CONNECT WITH USER01 IDENTIFIED BY AAA123456 using
'DMSERVER';    --DMSERVER 为实例名
```

例 2 使用 DM 数据库，创建一个连接到 IP 地址为 192.168.0.225 机器上的 oracle 数据库的外部链接。可以通过三种方式创建：一网络服务名 tsn_name；二连接描述符 description；三 <IP 地址>/<服务名>。

（一） 通过网络服务名创建

首先介绍 Oracle 网络服务名的配置方法。网络服务名配置成功才能创建 DBLINK

配置本地的 Oracle 网络服务名 ORCL，连接 225 机器上的 orcl 实例。详细步骤如下：



步骤 一



步骤 二



步骤 三



步骤 四

其次，创建 dblink。

网络服务名配置成功后，就可以使用网络服务名 ORCL 或网络连接描述符创建 DBLINK.

```
CREATE LINK LINK1 CONNECT 'ORACLE' WITH USER01 IDENTIFIED BY USER01 USING 'ORCL';
```

（二） 通过连接描述符创建

```
CREATE OR REPLACE LINK LINK1 CONNECT 'ORACLE' WITH USER01 IDENTIFIED BY USER01 USING '(DESCRIPTION =
    (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.225)(PORT = 1521)))
    (CONNECT_DATA = (SERVICE_NAME = orcl) )
  )';
```

（三） 通过<IP 地址>/<服务名>创建

```
CREATE LINK LINK1 CONNECT 'ORACLE' WITH USER01 IDENTIFIED BY USER01 USING '192.168.0.225/orcl';
```

16.2 删除外部链接

删除一个外部链接。

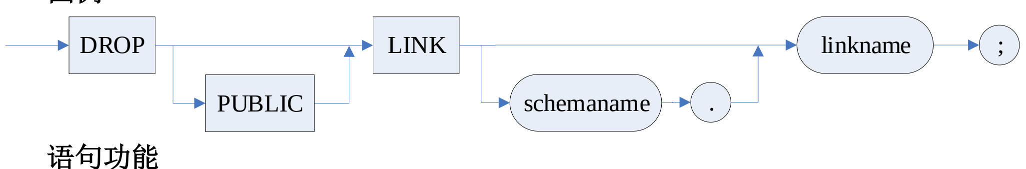
语法格式

```
DROP  [PUBLIC] LINK  [<模式名>.]<外部链接名>;
```

参数

1. <模式名> 指明被操作的外部链接属于哪个模式，缺省为当前模式；
2. <外部链接名> 指明被操作的外部链接的名称。

图例



语句功能

删除一个外部链接。

使用说明

只有链接对象的创建者和 DBA 拥有该对象的删除权限。

举例说明

删除外部链接 LINK1。

```
DROP LINK LINK1;
```

16.3 使用外部链接

通过外部链接，可以对远程服务器的对象进行查询或进行增删改操作，可以调用远程的过程。

使用外部链接进行查询或增删改的语法格式与普通格式基本一致，唯一的区别在于指定外部链接表时需要使用如下格式作为表或视图的引用：

```
[TABLENAME | VIEWNAME] [LINK | @] 链接名
```

举例说明

使用外部链接查询 LINK1 上的远程表进行查询

```
SELECT * FROM SYSOBJECTS LINK LINK1;
```

或对远程表进行插入数据：

```
INSERT INTO T1@LINK1 VALUES(1,2,3);
```

也可以查询本地表或其他链接的表对远程表进行操作，如

```
UPDATE T1@LINK1 SET C1 = C1+1 WHERE C2 NOT IN (SELECT ID FROM LOCAL_TABLE);
DELETE FROM T1@LINK1 WHERE C1 IN (SELECT ID FROM T2@LINK2);
```

使用外部链接，可以调用远程的存储过程，但是不支持调用远程的函数，使用中有以下约束：

（1）参数数据类型为 SQL 类型，不允许为 DMSQL 程序类型；

（2）参数数据类型不允许为复合类型。

其使用方式为：

```
[CALL] [<模式名>.][<包名>.]<过程名> [@] <外部链接名>(<参数列>);
```

使用限制

外部链接的使用有以下限制：

1. DM-DM 的同构外部链接不支持 MPP 环境，DM 与异构数据库的外部链接支持 MPP 环境；
2. 增删改不支持 INTO 语句；
3. 不支持使用游标进行增删改操作；
4. 不支持操作远程表的复合类型列；
5. DBLINK 理论上不支持 LOB 类型列的操作，但支持简单的增删改语句中使用常量来对 LOB 类型列进行操作。

另外，DM 连接异构数据库的外部链接还有如下使用限制：

1. 数据类型以 DM 为基础，不支持 DM 没有的数据类型；
2. 语法以 DM 的语法为标准，不支持 DM 不兼容的语法；
3. 主键更新，如果是涉及到多个服务器的语句，不能保证更新操作一定成功；
4. 使用 CREATE VIEW view_name(view_col_name) AS SELECT ITEM FROM T@LINK 方式创建的查询远程对象的本地视图，对于异构库，不能保证操作一定成功。对于查询异构库远程对象的本地视图，最好采用 CREATE VIEW AS SELECT ITEM AS alias_name FROM T@LINK 方式创建。

第 17 章 闪回查询

当系统 INI 参数 ENABLE_FLASHBACK 置为 1 时，闪回功能开启，可以进行闪回查询。MPP 环境不支持闪回查询。

17.1 闪回查询子句

闪回查询子句的语法，是在数据查询语句（参考第 4 章）的基础上，为 FROM 子句增加了闪回查询子句。

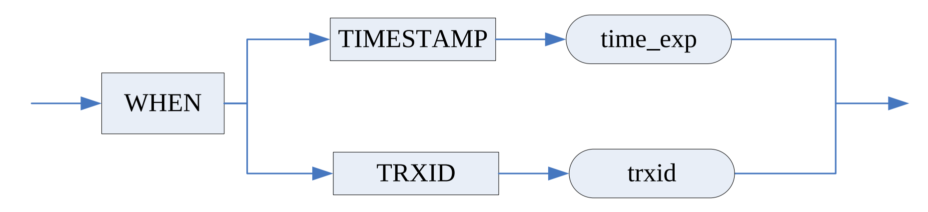
语法格式

```
<闪回查询子句>::=WHEN <TIMESTAMP time_exp> | <TRXID trxid>
```

参数

1. time_exp　一个日期表达式，一般用字符串方式表示
2. trxid　指定事务 ID 号

图例



语句功能

用户通过闪回查询子句，可以得到指定表过去某时刻的结果集。指定条件可以为时刻，或事务号。

使用说明

1. 闪回查询只支持普通表（包括加密表与压缩表）、临时表和堆表，不支持水平分区表、列存储表、外部表与视图；
2. 闪回查询中 trxid 的值，一般需要由闪回版本查询（见下节）的伪列来确定。实际使用中多采用指定时刻的方式。

举例说明

例 1 闪回查询特定时刻的 PERSON_TYPE 表。

查询 PERSON_TYPE 表。

```
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如下表所示。

PERSON_TYEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表

在 2012-01-01 12:22:49 时刻插入数据，并提交。

```
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('防损员');
```

```
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('保洁员');
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如下表所示。

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	防损员
6	保洁员

使用闪回查询取得 2012-01-01 12:22:45 时刻的数据。此时刻在插入数据的操作之前，可见此时的结果集不应该有 2012-01-01 12:22:49 时刻插入的数据。

```
SELECT * FROM PERSON.PERSON_TYPE WHEN TIMESTAMP '2012-01-01 12:22:45';
```

结果集如下表所示。

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表

在 2012-01-01 12:23:29 时刻删除数据，并提交。

```
DELETE FROM PERSON.PERSON_TYPE WHERE PERSON_TYPEID > 5;
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如下表所示。

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	防损员

使用闪回查询得到删除前的数据。

```
SELECT * FROM PERSON.PERSON_TYPE WHEN TIMESTAMP '2012-01-01 12:23:00';
```

结果集如下表所示。

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	防损员
6	保洁员

例 2 闪回查询指定 TRXID 的 PERSON_TYPE 表。

要获得 TRXID 信息，需要通过闪回版本查询的伪列 VERSIONS_ENDTRXID。（详细内

容见下节"闪回版本查询")

在 2012-01-01 12:24:05 时刻修改数据，并提交。

```
UPDATE PERSON.PERSON_TYPE SET NAME='保安员' WHERE PERSON_TYPEID=5;
COMMIT;
UPDATE PERSON.PERSON_TYPE SET NAME='收银员' WHERE PERSON_TYPEID=5;
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如下表所示。

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	收银员

进行闪回版本查询，确定 TRXID。

```
SELECT VERSIONS_ENDTRXID, NAME FROM PERSON.PERSON_TYPE VERSIONS BETWEEN
TIMESTAMP '2012-01-01 12:24:00' AND SYSDATE;
```

得到结果集如下表所示。

VERSION_ENDTRXID	NAME
NULL	采购经理
NULL	采购代表
NULL	销售经理
NULL	销售代表
323	收银员
322	保安员
NULL	防损员

根据 TRXID 确定版本。

```
SELECT * FROM PERSON.PERSON_TYPE WHEN TRXID 322;
```

结果集如下表所示。

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	保安员

17.2 闪回版本查询

语法格式

```
<闪回版本查询子句>::=VERSIONS BETWEEN <TIMESTAMP time_exp1 AND time_exp2> |
<TRXID trxid1 AND trxid2>
```

参数

1. time_exp 日期表达式，一般用字符串方式表示。time_exp1 表示起始时间，

time_exp2 表示结束时间

2. trxid 指定事务 ID 号，整数表示。trxid1 表示起始 trxid，trxid2 表示结束 trxid

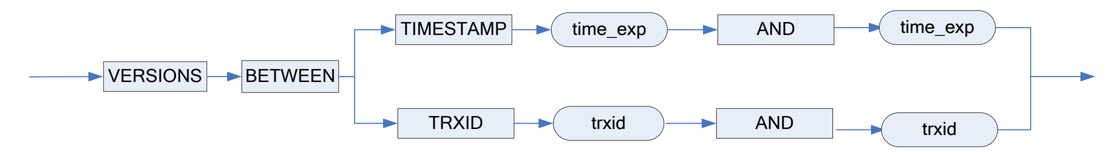
使用说明

1. 闪回版本查询支持普通表（包括加密表与压缩表）、临时表和堆表，不支持水平分区表、列存储表、外部表与视图；
2. 支持如表 17.2.1 所示伪列，作为闪回版本查询的辅助信息。

表 17.2.1 闪回版本查询支持的伪列

伪列	说明
VERSIONS_START{TRXID|TIME}	起始 TRXID 或时间戳
VERSIONS_END{TRXID|TIME}	提交 TRXID 或时间戳。如果该值为 NULL，表示行版本仍然是当前版本
VERSIONS_OPERATION	在行上的操作（I=Insert,D=Delete,U=Update）

图例



语句功能

用户通过闪回版本查询子句，可以得到指定表过去某个时间段内，事务导致记录变化的全部记录。指定条件可以为时刻，或事务号。

举例说明

例 1 闪回版本查询指定时间段内，PERSON_TYPE 表的记录变化。

在 2012-01-01 12:24:05 时刻修改数据，并提交。

```
UPDATE PERSON.PERSON_TYPE SET NAME='保安员' WHERE PERSON_TYPEID=5;
COMMIT;
UPDATE PERSON.PERSON_TYPE SET NAME='收银员' WHERE PERSON_TYPEID=5;
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如下表所示。

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	收银员

进行闪回版本查询，获得指定时间段内变化的记录。

```
SELECT VERSIONS_ENDTRXID, NAME FROM PERSON.PERSON_TYPE VERSIONS BETWEEN
TIMESTAMP '2012-01-01 12:24:00' AND SYSDATE;
```

得到结果集如下表所示。

VERSION_ENDTRXID	NAME
NULL	采购经理
NULL	采购代表

NULL	销售经理
NULL	销售代表
323	收银员
322	保安员
NULL	防损员

17.3 闪回事务查询

闪回事务查询提供系统视图 V$FLASHBACK_TRX_INFO 供用户查看在事务级对数据库所做的更改。根据视图信息，可以确定如何还原指定事务或指定时间段内的修改。

使用说明

系统视图名为 V$FLASHBACK_TRX_INFO，定义如表 17.3.1 所示。

表 17.3.1

列名	数据类型	说明
START_TRXID	BIGINT	事务中第一个 DML 的 TRXID
START_TIMESTAMP	TIMESTAMP	事务中第一个 DML 的时间戳
COMMIT_TRXID	BIGINT	提交事务的 TRXID
COMMIT_TIMESTAMP	TIMESTAMP	提交事务时的时间戳
LOGON_USER	VARCHAR(256)	拥有事务的用户
UNDO_CHANGE#	INT	记录修改顺序序号
OPERATION	CHAR(1)	DML 操作类型。 D：删除；U：修改；I：插入；N：更新插入（专门针对 CLUSTER PRIMARY KEY 的插入）； C：事务提交；P：预提交记录；O：default
TABLE_NAME	VARCHAR(256)	DML 修改的表
TABLE_OWNER	VARCHAR(256)	DML 修改表的拥有者
ROW_ID	BIGINT	DML 修改行的 ROWID
UNDO_SQL	VARCHAR(3900)	撤销 DML 操作的 SQL 语句

举例说明

例 1 查询指定时间之后的事务信息，可为闪回查询操作提供参考。

```
SELECT * FROM V$FLASHBACK_TRX_INFO WHERE COMMIT_TIMESTAMP > '2012-01-01 12:00:00';
```

第 18 章 JSON

JSON（JavaScript Object Notation）是完全独立于语言的文本格式，是一种轻量级的数据交换格式。

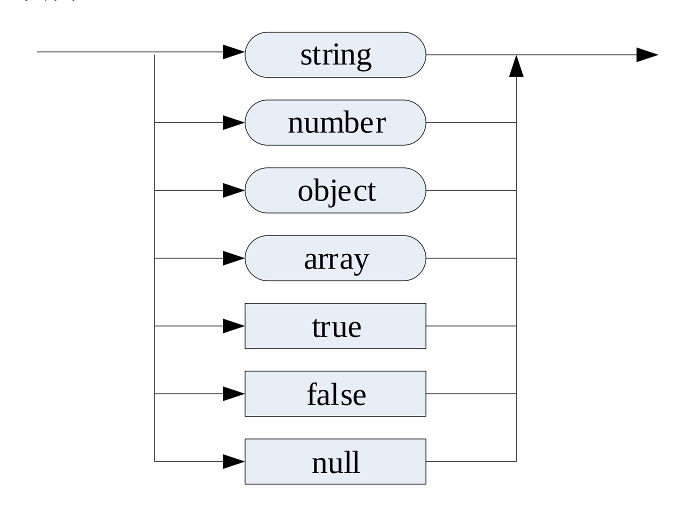
DM 数据库支持对 JSON 数据进行存储和查询。在 DM 数据库中 JSON 数据以字符串形式存储。DM 建议用户在插入 JSON 数据之前，使用 IS JSON 来验证输入 JSON 数据的正确性。

本节中的例子，除了特殊说明之外，建表语句都请参考 18.5 一个简单的例子。

18.1 数据类型

JSON 支持的数据类型包括：字符串 string、数字 number、布尔值 true 和 false、null、对象 object 和数组 array。JSON 的各种数据类型可以嵌套使用。

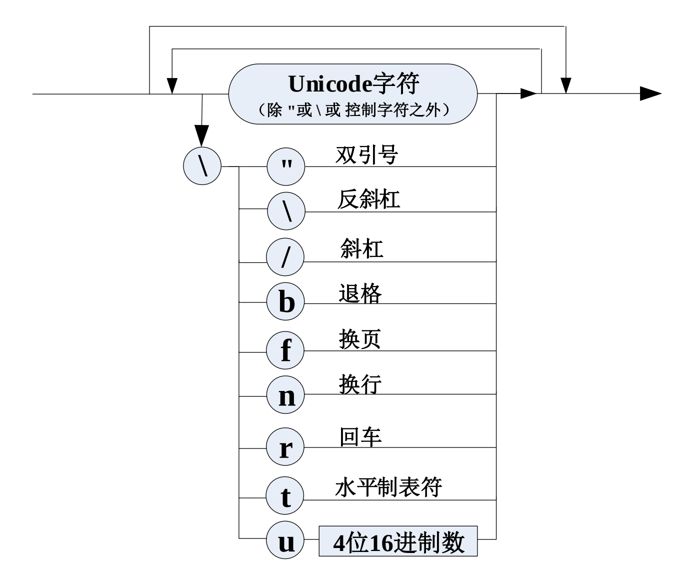
图例



18.1.1 string

字符串长度需要大于等于 0，可以为空串。一般建议使用引号将 string 括起，DM 支持双引号和单引号。string 作为名称，在 IS JSON(LAX)情况下可以不用引号，但作为值时必须使用引号。语法图中第一个反斜杠"\"表示转义字符。

图例



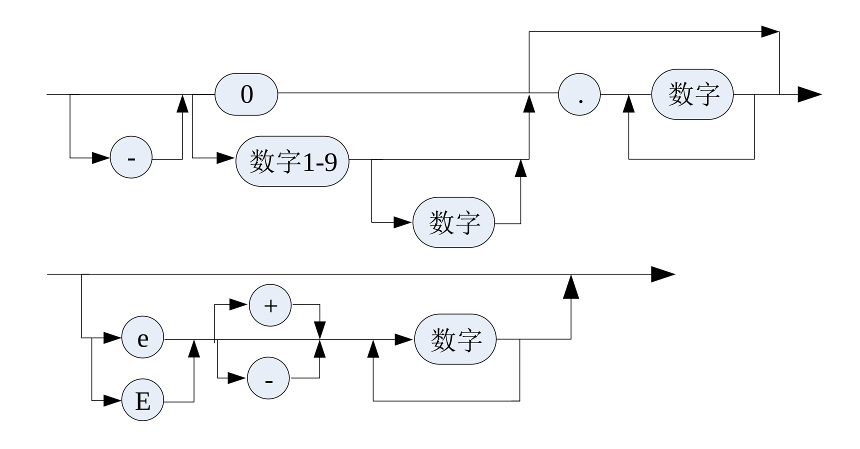
使用说明

字符串必须以 a-z 或 A-Z 开始，后续字符可以包含 0-9；如果不遵守这个规则或者包含其他字符，则必须以引号括起。只有空格能够以“”的形式出现在字符串中，其他特殊字符则不可以。

18.1.2 number

数字不支持八进制和十六进制。

图例



18.1.3 true、false

true和false代表布尔值，使用时不需要加引号。一般作为值，但也可以作为名称。

插入JSON列时，必须注意以下2点：

1. true和false作为值。IS JSON (STRICT)时必须是小写，否则报错；IS JSON (LAX)时，则不区分大小写，如：TRUE、True、tRue均是合法的。但查询语句中都必须以小写形式返回值。

2. true和false作为名称。这是一种特殊的用法，一般不建议这样用。仅在IS JSON (LAX)时支持，否则报错。因为用法特殊，不管在名称中大小写与否，查询语句中都只有小写才能返回对应的值。

举例说明

```
drop table t_json_boolean CASCADE;
create table t_json_boolean(c1 int,
c2 varchar2(100) CHECK (c2 IS JSON(STRICT)),
c3 varchar2(100) CHECK (c3 IS JSON(LAX))
);

insert into t_json_boolean values(1,'{"dameng":TRUE}',NULL);
--违反 CHECK 约束，需要将 TRUE 修改为 true

insert into t_json_boolean values(2,NULL,'{"dameng":TRUE}');
--插入成功，LAX 不区分大小写

insert into t_json_boolean values(3,'{true:1}',NULL);
--违反 CHECK 约束，STRICT 时 true 和 false 不能作为名称

insert into t_json_boolean values(4,NULL,'{TRUE:1}');
--插入成功，LAX 时 true 和 false 可以作为名称且不区分大小写

commit;

select C1,json_value(c3, '$.dameng') from t_json_boolean
--运行结果：
2          true
4          NULL

select C1,json_value(c3, '$.dameng' returning number) from t_json_boolean;
--运行结果：
2          1
4          NULL

select C1,json_value(c3, '$.true') from t_json_boolean;
--运行结果：
2          NULL
4          1

select C1,json_value(c3, '$.TRUE') from t_json_boolean;
--运行结果：
2          NULL
4          NULL
```

18.1.4 null

null代表JSON数据为空，它与SQL语句中的值为NULL是不同的。null使用时不需要加引号，一般作为值，但也可以作为名称。

插入JSON列时，必须注意以下3点：

1. null作为值。IS JSON (STRICT)时必须是小写，否则报错；IS JSON (LAX)时，则不区分大小写，如：NULL、nUll、nULL均是合法的。

2. json_value时，null以SQL语句NULL的形式返回，此时无法区分是SQL的NULL还是json数据的null；json_query时，null返回必须指定WITH WRAPPER的形式返回，如[null]，查询语句中必须以小写形式才能有返回值。

3. null作为名称。这是一种特殊的用法，一般不建议这样用。仅在IS JSON (LAX)时支持，否则报错。因为用法特殊，不管名称中的大小写，查询语句中都只有小写才能返回对应的值。

从上可以看出，null的使用规则(1)(3)与true、false的使用规则基本一致。

举例说明

```
drop table t_json_null;
create table t_json_null(c1 int,
c2 varchar2(100) CHECK (c2 is json));

insert into t_json_null values(1,null);--SQL 语句的 null
insert into t_json_null values(2,'{"dameng":null}');--json 数据的 null
insert into t_json_null values(3,NULL);--SQL 语句的 null
insert into t_json_null values(4,'{"dameng":NULL}');--json 数据的 null
commit;

SQL> select json_value(c2, '$.dameng') from t_json_null;

行号       JSON_VALUE(C2,'$.dameng')
---------- -------------------------
1          NULL
2          NULL
3          NULL
4          NULL
--第 2、4 行全部转化为 SQL 的 NULL。和第 1、2 行一样。

SQL> select json_query(c2, '$.dameng' WITH WRAPPER) from t_json_null;

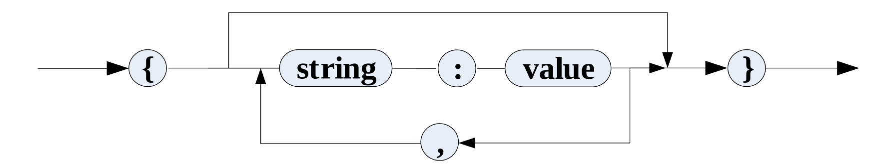
行号       json_query(C2,'$.dameng'WITHWRAPPER)
---------- ------------------------------------
1          NULL
2          [null]
3          NULL
4          [null]
```

```
--查询可以看出四行数据的不同。2、4 行为 json 数据。
```

18.1.5 object

对象以"{"开始，"}"结束。名称/值对的书写格式：<string>: <value>。

语法格式

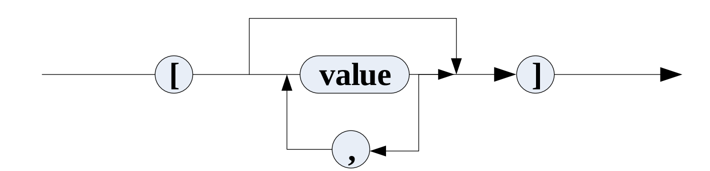


<value>：可以是 JSON 数据类型中的任何一种。

18.1.6 array

数组是<值>的有序集合，数组在[]中书写。

语法格式



<value>：可以有 0 或多个 value，value 可以是数据类型中的任何一种。

18.2 函数

18.2.1 json_value

json_value 函数的返回值必须是单值且是标量数据类型。

语法格式

```
<json_value 函数> ::= JSON_VALUE(<参数 1> , <参数 2>
                [<RETURNING 项>] [ASCII] [<ERROR 项>])
<参数 1> ::=  <列名>|JSON 数据
<参数 2> ::=  <路径表达式>
<RETURNING 项> :: = RETURNING <json 数据类型>
<json 数据类型>:: =string|number|true|false|null
<ERROR 项> :: =  <NULL | ERROR | <DEFAULT 项>> ON ERROR
<DEFAULT 项> :: = DEFAULT  '<value>'
```

参数

<参数 1>：必须是 json 数据类型的对象 object、数组 array 或 NULL（注意：sql 的 NULL，非 json 的 null），否则函数报错。

<参数 2>：路径表达式，请参考 18.2.3.1 节。

<RETURNING 项>：返回值类型，默认为 VARCHAR(8188)。出错时默认为 NULL ON ERROR;

<DEFAULT 项>:返回值必须与<RETURNING 项>中定义的类型匹配。当 RETURNING 字符串时，默认值可以是数字或字符串。布尔类型有两种表示方法：一是返回'true'和'false'（VARCHAR 类型），二是返回 1 和 0（NUMBER 类型）。

使用说明

(1) 数据类型中的 string 表示变长字符串，具体包括：varchar，varchar2，clob，varbinary 等，可以指定精度，如：varchar(100)。

(2) 数据类型中的 number 对应 dm 的 dec 类型，可以有多种写法，如：decimal，也可以指写精度，例如:dec(10,5)。

(3) true 和 false 在指定 RETURNING number 时返回值对应 1 和 0。

举例说明：

1．使用 json_value 查询对象的值。

```
SQL> SELECT ID,json_value(po_document, '$.Requestor') FROM j_purchaseorder;


行号      ID         JSON_VALUE(PO_DOCUMENT,'$.Requestor')
----------------- ----------------------------------------------------
1         1          Alexis Bull
```

2．使用 json_value 查询数组的值。

```
SQL> SELECT ID,json_value(po_document,
'$.ShippingInstructions.Phone[1].type') FROM j_purchaseorder;


行号   ID  JSON_VALUE(PO_DOCUMENT,'$.ShippingInstructions.Phone[1].type')
---------------------------------------------------------------------------
1      1          Mobile
```

18.2.2 json_query

json_query 的返回结果是一个或多个 JSON 数据。多值返回时必须指定 WITH WRAPPER。单值返回时，标量类型必须指定 WITH WRAPPER，object 或 array 则不需要。

语法格式

```
<json_query 函数> ::= json_query(<参数 1>,<参数 2> [<RETURNING 项>] [PRETTY] [ASCII]
[<WRAPPER 项>] [<ERROR 项>])
<参数 1> ::=  <列名>| JSON 数据
<参数 2> ::=  <路径表达式>
<RETURNING 项> :: = RETURNING <json 数据类型>
<json 数据类型>:: =string| true| false| null
<WRAPPER 项> :: =
                WITH[CONDITIONAL|UNCONDITIONAL][ARRAY] WRAPPER|
                WITHOUT[ARRAY]WRAPPER
<ERROR 项> :: =  < NULL | ERROR | EMPTY>  ON ERROR
```

参数

<参数 1>： 必须是 json 数据类型，否则函数报错。具体数据类型，请参考 18.1 节。

<参数 2>： 路径表达式，请参考 18.2.3.1 节。

<RETURNING 项>：返回值类型，默认为 VARCHAR(8188)。出错时默认为 NULL ON ERROR;

<数据类型>:使用方法请参考 json_value。

举例说明

1. 使用 json_query，以数组的形式返回 JSON 数据。

```
SQL>  select json_query(po_document,'$.PONumber' returning varchar2 with wrapper) FROM j_purchaseorder;

行号      json_query(PO_DOCUMENT,'$.PONumber'WITHWRAPPER)
-----------------------------------------------------------------------
1         [1600]
```

18.2.3 函数参数详解

18.2.3.1 路径表达式

JSON 数据的查询需要使用路径表达式。路径表达式为通配符"*"和 array 范围。使用规则如下：

1. object 和 array 必须以"$"开始；
2. object 紧跟"."，则表明对象是 object，且需要指定<名称>；
3. object 紧跟通配符"*"，则表示 object 的所有<名称>；
4. array 紧跟"["，以"]"结束，可以使用通配符"[*]"查找数组所有<值>；
5. array 索引可以是 0, 1, 2,…，起始值为 0；
6. array 中的范围必须非递减，例如：[3, 8 to 10, 12]、[1,2,3,3 to 3,3]；
7. 对于不是 array 的数据，$[0]表示本身。

例如：查询 19.6 节例子中数据 Mobile 值，使用的路径表达式为：

$.ShippingInstructions.Phone[1].type。

18.2.3.2 PRETTY 和 ASCII

PRETTY 以缩进的形式显示字符，ASCII 以\uXXXX 十六进制的形式显示非 Unicode 字符，使用原则如下：

1. 默认为非 PRETTY；
2. 两者一起使用时，PRETTY 必须在 ASCII 之前；
3. json_value 和 json_query 都可以使用 ASCII；
4. 只有 json_query 可以使用 PRETTY。

举例说明：

```
SQL>SET KEEPDATA ON
--返回结果中的换行符不用空格替代

SQL> SELECT json_query(po_document,'$.ShippingInstructions'    RETURNING VARCHAR PRETTY WITH WRAPPER ERROR ON ERROR) from j_purchaseorder;

json_query(PO_DOCUMENT,'$.ShippingInstructions'PRETTYWITHWRAPPERERROR
```

```
ONERROR)
---------- ---------------------------------------------------------------
[
  {
    "name" : "Alexis Bull",
    "Address" :
    {
      "street" : "200 Sporting Green",
      "city" : "South San Francisco",
      "state" : "CA",
      "zipCode" : 99236,
      "country" : "United States of America"
    },
    "Phone" :
    [
      {
        "type" : "Office",
        "number" : "909-555-7307"
      },
      {
        "type" : "Mobile",
        "number" : "415-555-1234"
      }
    ]
  }
]
```

18.2.3.3 WRAPPER 项

只有 json_query 可以使用<WRAPPER 项>。缺省为 WITHOUT WRAPPER。

- WITH WRAPPER：以 array 的形式返回字符串，显示匹配路径表达式下的所有 JSON 数据，array 元素的顺序不是固定的。
- WITHOUT WRAPPER：只返回匹配路径表达式的单个 JSON object 或 array。如果是标量类型（非 object 或 array）或多于 1 条数据则报错返回。
- WITH CONDITIONAL WRAPPER：单个 JSON object 或 array 时，等价于 WITHOUT WRAPPER；其他情况等价于 WITH WRAPPER。
- WITH UNCONDITIONAL WRAPPER 和 WITH WRAPPER 是等价的。
- ARRAY 关键字可以省略，省略和不省略意义一样。

表 对比<WRAPPER 项>中不同组合情况

路径表达式	WITH WRAPPER	WITHOUT WRAPPER	WITH CONDITIONALWRAPPER
{"id": 38327}	**[**{"id": 38327}**]**	{"id": 38327}	{"id": 38327}
[42, "a", true]	**[**[42, "a", true]**]**	[42, "a", true]	[42, "a", true]
42	**[**42**]**	**Error**	**[**42**]**
42, "a", true	**[**42, "a", true**]**	**Error**	**[**42, "a", true**]**

none	[]	**Error**	[]

举例说明

使用 WITH WRAPPER 关键字，以数组的形式返回查询结果。

```
    SELECT json_query(po_document, '$.ShippingInstructions.Phone[*].type' WITH WRAPPER) FROM j_purchaseorder;
    --返回结果：
    ["Office","Mobile"]
```

18.2.3.4 ERROR 项

JSON 函数和条件表达式的错误处理。只有路径表达式语法正确时，ERROR 语句才有效。

- ERROR ON ERROR：出错时则返回该错误；
- NULL ON ERROR：出错时返回 NULL；
- EMPTY ON ERROR：出错时返回[]，只有 json_query 可以用；
- DEFAULT '<value>' ON ERROR：出错时返回指定的值，且<value>必须是常量，只有 json_value 可用。

举例说明

展示<ERROR 项>的使用效果。

```
    SELECT JSON_VALUE('[1,2]','$[0,1]' ERROR ON ERROR) FROM DUAL;
    --报错:JSON_VALUE 求值为多个值
    SELECT JSON_VALUE('[[1]]','$[0]' ERROR ON ERROR) FROM DUAL;
    --报错:JSON_VALUE 的计算结果为非标量值
    SELECT JSON_VALUE('[1]','$[1]' RETURNING VARCHAR DEFAULT 1 ON ERROR) FROM DUAL;
    --报错：默认值不匹配在 RETURNING 子句中定义的类型
    SELECT JSON_VALUE('[aa]','$[0]' RETURNING number default '1' on error) FROM DUAL;
    -------------------------------------------------------------------
    1
```

18.3 使用 IS JSON/IS NOT JSON 条件

IS JSON/IS NOT JSON 条件，用于判断 JSON 数据合法性。当判断语法正确时，IS JSON 返回 true，IS NOT JSON 返回 false。

语法格式

```
    <IS_JSON_clause>::=
        IS [NOT] JSON [(STRICT|LAX)] [<unique_clause>]
    <unique_clause>::=
        WITH UNIQUE KEYS |
        WITHOUT UNIQUE KEYS
```

详细的参数介绍如下：

■ IS JSON/IS NOT JSON

通常，IS JSON/IS NOT JSON 条件被用于 CHECK 约束中。当对 JSON 数据使用 IS JSON/IS NOT JSON 的 CHECK 约束时，在插入过程中会相对慢一些。当能够保证 JSON 数据的合法性时，可以 DISABLE 这个约束，建议不要 DROP 该约束。

举例说明

1.在 CHECK 中使用 IS JSON，保证插入的数据，符合 JSON 标准。

```
drop table json_is_json cascade;
CREATE TABLE json_is_json
(id int NOT NULL,
po_document CLOB
CONSTRAINT ensure_json_unique CHECK (po_document IS JSON));
--创建成功
INSERT INTO json_is_json VALUES (1,'{"PONumber" : 1600, "PONumber" : 1800}');
--插入成功
INSERT INTO json_is_json VALUES (2,'OK');  --不是 JSON 数据，报错违反 CHECK 约束
INSERT INTO json_is_json VALUES (3,NULL);--IS JSON 可以成功插入 NULL，IS NOT JSON 时也可以成功插入 NULL。
```

2.在插入语句中使用 IS JSON，保证从其他表中拷入的数据，是符合 JSON 标准的。

```
drop table j_purchaseorder_insert;
--第一步：创建表
CREATE TABLE j_purchaseorder_insert
(c1int NOT NULL,
c2 TIMESTAMP (6) WITH TIME ZONE,
c3 VARCHAR);
--第二步：插入数据
INSERT INTO j_purchaseorder_insert select id, date_loaded, po_document from j_purchaseorder where po_document IS JSON;
```

■ **LAX/STRICT**

LAX/STRICT 用来规范 JSON 数据格式。STRICT 数据格式比 LAX 要求更严格。默认是 LAX。详细规则如下：

1.STRICT 和 LAX 时，true/false/null 大小写要求不同，详情请参考 true/false、null 章节。

2.IS JSON(STRICT)时，正数不能以"+"或"."开头，不能有前导 0，不能以"."结尾；LAX 时则可以。

3. IS JSON(STRICT)时，object 的 string:value 链表或 array 的 value 链表后不能多追加","；IS JSON(LAX)时则可以。

4.JSON 函数不区分 STRICT 和 LAX。

举例说明

将同一组数据插入到分别使用了 LAX/STRICT 的表中，对比区别。

第一步，在表 t_json_s 中使用 IS JSON (STRICT)，表 t_json_l 中使用 IS JSON (LAX)。

```
drop table t_json_s;
drop table t_json_l;
create table t_json_s(c1 int, c2 varchar2(100) constraint c2_json CHECK (c2 IS JSON (STRICT)));
create table t_json_l(c1 int, c2 varchar2(100) constraint l_c2_json CHECK (c2 IS JSON (LAX)));
```

第二步，分别插入上述数据。

```
insert into t_json_l values(1,'{''dmdatabase'':29}'); --<名称>：没有使用双引号
insert into t_json_l values(2,'{"2dmdatabase":29}'); -- 正确
insert into t_json_l values(3,'{"dmdatabase":.29}'); -- <值>：缺失整数部分
insert into t_json_l values(4,'{"dmdatabase":NULL}'); -- <值>：NULL 没有小写
insert into t_json_l values(5,'{"dmdatabase":False}'); -- <值>：False 没有小写
insert into t_json_l values(6,'{"dmdatabase":29,"dmdatabase":30}'); -- 正确
insert into t_json_l values(7,'{"dm data base":29}'); -- 正确
insert into t_json_l values(8,'{"dmdatabase":dmdatabase}');--<值>：字符串缺双引号
insert into t_json_l values(9,'{dmdatabase: "dmdatabase"}'); --<值>：字符串前有空格
insert into t_json_l values(10,'{"dmdatabase":2s}'); -- <值>：字符串没有加双引号
insert into t_json_l values(11,'{"dmdatabase":''2s''}'); -- <值>：字符串缺双引号
insert into t_json_l values(12,'{dmdatabase:29 }'); -- <值>：数字后有空格
```

第三步，查询两个表中数据，对比插入结果。

```
SQL> select * from t_json_l order by c1;

行号       C1          C2
---------- ----------- ---------------------------------
1          1           {'dmdatabase':29}
2          2           {"2dmdatabase":29}
3          3           {"dmdatabase":.29}
4          4           {"dmdatabase":NULL}
5          5           {"dmdatabase":False}
6          6           {"dmdatabase":29,"dmdatabase":30}
7          7           {"dm data base":29}
8          9           {dmdatabase: "dmdatabase"}
9          11          {"dmdatabase":'2s'}
10         12          {dmdatabase:29 }

10 rows got
--返回结果：只有第 8 行和第 10 行插入失败。

 SQL> select * from t_json_s order by c1;

行号       C1          C2
---------- ----------- ---------------------------------
1          2           {"2dmdatabase":29}
2          6           {"dmdatabase":29,"dmdatabase":30}
3          7           {"dm data base":29}
--只有第 2、6、7 行插入成功。其他没有插入成功的。插入时均报错：违反 CHECK 约束。
```

- **WITH UNIQUE KEYS/ WITHOUT UNIQUE KEYS**

与 IS JSON 一起使用；

使用 WITH UNIQUE 时，对象中不可以有同名的名称，即名称必须唯一；

使用 WITHOUT UNIQUE 时，对象中可以有同名的名称，但是查询时只会随机选择其中一个，DM 默认选择第一个。

缺省为使用 WITHOUT UNIQUE。

举例说明

在 CHECK 约束中使用 IS JSON WITH UNIQUE KEYS，保证插入的数据没有重复。

```
DROP TABLE json_unique CASCADE;
CREATE TABLE json_unique
(id int NOT NULL,
date_loaded TIMESTAMP (6) WITH TIME ZONE,
po_document CLOB
CONSTRAINT ensure_json_unique CHECK (po_document IS JSON WITH UNIQUE KEYS));
--创建成功
INSERT INTO json_unique VALUES (
111,SYSTIMESTAMP,'{"PONumber" : 1600, "PONumber" : 1800}');
--报错：违反 CHECK 约束条件 (ENSURE_JSON_UNIQUE)
```

18.4 视图

JSON 数据信息都存储在 DBA_JSON_COLUMNS、USER_JSON_COLUMNS 和 ALL_JSON_COLUMNS 视图中，下面进行详细介绍。

18.4.1 视图使用说明

1. 当 JSON 列的 IS JSON 约束被失效后，该列仍然在视图中显示。

2. 当 IS JSON 涉及多列时，则所有涉及的列均在视图中显示。例如：c1||c2 is json，则 c1 和 c2 列均在视图中显示。

3. 如果 IS JSON 与其他约束进行”与”(AND)运算时，则所有涉及的列均在视图中显示。例如：c1 = '1' and c2 is json。

4. 如果 IS JSON 与其他约束进行”或”(OR)运算时，则所有列均不在视图中显示。例如：c1 is json OR c2 < 1000，即使是 c1 is json OR c2 is json 也不行。

5. 如果 IS NOT JSON 作为 CHECK 约束时，则该列不在视图中显示。例如：C1 is not json。同理:c1||c2 is not json，则 c1 和 c2 均不能在下列视图中显示。

6. 如果 NOT IS JSON 作为 CHECK 约束时，则该列也不能在下列视图中显示。例如：not(c2 is json)。

7. 当虚拟列相关的实际列使用 IS JSON 作为 CHECK 约束时，该虚拟列不在视图中显示；当虚拟列使用 IS JSON 作为 CHECK 约束时，仅该虚拟列在视图中显示，实际列则不在视图中显示。

18.4.2 DBA_JSON_COLUMNS

显示数据库中所有的 JSON 数据信息。

列名	数据类型	说明
OWNER	VARCHAR(128)	模式名
TABLE_NAME	VARCHAR(128)	表名
COLUMN_NAME	VARCHAR(128)	列名
FORMAT	VARCHAR(4)	格式化。统一为 TEXT
DATA_TYPE	VARCHAR(11)	列的数据类型。可能的取值：VARCHAR2、CLOB、LONGVARCHAR、TEXT、UNDEFINED(对于 CHAR 类型、VARCHAR 类型)

18.4.3 USER_JSON_COLUMNS

显示当前用户所拥有的 JSON 数据信息。该视图比 DBA_JSON_COLUMNS 视图少了一列 OWNER。

18.4.4 ALL_JSON_COLUMNS

显示当前用户有权访问的 JSON 数据信息。该视图列与 DBA_JSON_COLUMNS 完全相同。

18.5 一个简单的例子

在数据库表中插入 JSON 数据，并查询它。

第一步，在 CHECK 中使用 IS JSON，保证插入的数据符合 JSON 标准格式。

```
drop table j_purchaseorder;
--创建表
CREATE TABLE j_purchaseorder
(id int NOT NULL,
date_loaded TIMESTAMP (6) WITH TIME ZONE,
po_document VARCHAR
CONSTRAINT ensure_json CHECK (po_document IS JSON));

--插入数据。插入 po_document 列的就是 JSON 数据，也可以使用 dmfldr 工具导入数据。
INSERT INTO j_purchaseorder VALUES (1,SYSTIMESTAMP,'{
"PONumber" : 1600,
"Reference" : "ABULL-20140421",
"Requestor" : "Alexis Bull",
"User" : "ABULL",
"CostCenter" : "A50",
"ShippingInstructions" : {"name" : "Alexis Bull",
        "Address": {"street" : "200 Sporting Green",
        "city" : "South San Francisco",
        "state" : "CA",
        "zipCode" : 99236,
```

```
          "country" : "United States of America"},
          "Phone" : [{"type" : "Office", "number" : "909-555-7307"},
                    {"type" : "Mobile", "number" : "415-555-1234"}]
          },
    "Special Instructions" : null,
    "AllowPartialShipment" : true,
    "LineItems" : [{"ItemNumber" : 1,
                  "Part" : {"Description" : "One Magic Christmas",
                            "UnitPrice" : 19.95,
                            "UPCCode" : 13131092899},
                  "Quantity" : 9.0},
               {"ItemNumber" : 2,
               "Part" : {"Description" : "Lethal Weapon",
                      "UnitPrice" : 19.95,
                      "UPCCode" : 85391628927},
               "Quantity" : 5.0}]
    }
    ');
```

查询结果：

```
    SQL> SELECT json_query(po_document,'$.*'   RETURNING VARCHAR PRETTY WITH
WRAPPER ERROR ON ERROR) from j_purchaseorder;

    行号  json_query(PO_DOCUMENT,'$.*'PRETTYWITHWRAPPERERRORONERROR)
    -------------------------------------------------------------------------
    1       [
         1600,
         "ABULL-20140421",
         "Alexis Bull",
         "ABULL",
         "A50",
         {
           "name" : "Alexis Bull",
           "Address" :
           {
             "street" : "200 Sporting Green",
             "city" : "South San Francisco",
             "state" : "CA",
             "zipCode" : 99236,
             "country" : "United States of America"
           },
           "Phone" :
           [
             {
               "type" : "Office",
```

```
          "number" : "909-555-7307"
        },
        {
          "type" : "Mobile",
          "number" : "415-555-1234"
        }
      ]
    },
  null,
  true,
    [
      {
        "ItemNumber" : 1,
        "Part" :
        {
          "Description" : "One Magic Christmas",
          "UnitPrice" : 19.95,
          "UPCCode" : 13131092899
        },
        "Quantity" : 9.0
      },
      {
        "ItemNumber" : 2,
        "Part" :
        {
          "Description" : "Lethal Weapon",
          "UnitPrice" : 19.95,
          "UPCCode" : 85391628927
        },
        "Quantity" : 5.0
      }
    ]
  ]
```

第 19 章 高级日志

19.1简介

行表和 HUGE 表在增删改查性能上存在差异，因此在实际的生产环境中，用户可能会同时使用一个行表来管理数据和一个 HUGE 表来分析数据。具体做法是对行表进行增删改操作，然后把行表中的数据复制到 HUGE 表中用于查询或分析。如果每次分析数据时都对行表进行全表查询插入 HUGE 表，性能较低。

为此提出一种解决方案：给行表添加日志辅助表用于记录行表的增删改和 TRUNCATE 操作，可以根据日志表实现对 HUGE 表的增量更新，以此来提高从行表复制数据到 HUGE 表的性能。

19.2使用须知

增量更新过程，我们只提供日志的记录以及日志记录规则的制定，真正执行增量更新是由用户根据日志记录自行操作。辅助表中登记信息，为某一时间点后源表数据的增量变化信息登记。

19.3语法

19.3.1管理日志辅助表

19.3.1.1 创建日志辅助表

创建日志辅助表，有两种方式：一是建表时创建；二是修改表时创建。

1. 建表时候使用<高级日志子句>创建日志辅助表

语法格式

```
CREATE  TABLE <表名定义> <表结构定义>;
<表名定义> ::= [<模式名>.] <表名>
<表结构定义>::=<表结构定义 1> | <表结构定义 2>
<表结构定义 1>::= (<列定义> {,<列定义>} [,<表级约束定义>{,<表级约束定义>}]) [ON COMMIT <DELETE | PRESERVE> ROWS] [<PARTITION 子句>][<空间限制子句>] [<STORAGE 子句>][<压缩子句>] [<ROW MOVEMENT 子句>][<高级日志子句>] [<add_log 子句>] [<DISTRIBUTE 子句>]
……
<高级日志子句>::= WITH ADVANCED LOG
省略号（……）部分请参考《DM8 Sql 语言使用手册》3.6.1.1 定义数据库基表
```

2. 修改表时使用<高级日志子句>添加日志表

语法格式

```
ALTER TABLE <高级日志子句>;
```

19.3.1.2 删除日志辅助表

语法格式

```
ALTER TABLE <TABLE_NAME> WITHOUT ADVANCED LOG;
```

19.3.1.3 删除日志辅助表的数据

语法格式

```
ALTER TABLE <TABLE_NAME> TRUNCATE ADVANCED  LOG;
```

数据清除后可能导致源表和 HUGE 无法同步，需慎重操作。

19.3.2使用日志辅助表的规则与约束

日志辅助表命名为"表名$ALOG"，用于记录源表的操作但不涉及具体数据。

规则与约束：

1. 每个源表仅支持设置一个日志辅助表。
2. 表删除的同时删除其日志辅助表。
3. 表更名时，日志表同步更名。
4. 由于其日志表名长度不得超过 128，因此表名长度不得超过 123。
5. 辅助表仅登记源表相关增删改及 TRUNCATE 等涉及数据变化的操作，却不涉及具体数据。
6. 源表执行 ADD/DROP/MODIFY COLUMN 的 DDL 操作时，也必须保证日志辅助表为空。
7. 如果表设置了高级日志功能，禁止或者不建议以下操作：
   1) 禁止对源表创建聚集索引
   2) 禁止删除源表上本存在的聚集索引
   3) 禁止直接对分区表的子表执行 DELETE、UPDATE、INSERT 以及 TRUNCATE
   4) 禁止在 ALTER TABLE 时，新建、删除或者修改主键，使主键失效或者生效，或者删除主键列
   5) 禁止对临时表、HUGE 表和间隔分区表设置高级日志表，禁止查询插入建表方式设置高级日志表。
   6) 禁止直接删除高级日志表以及创建后缀为"$ALOG"的表
   7) 禁止合并分区
   8) 禁止对表加列、删除列和修改列，禁止添加、分裂、交换和删除分区。交换分区时的普通表也禁止带有高级日志
   9) 表备份还原后无法控制数据跟踪，无法保证同步数据的正确性。因此不建议对该表进行备份还原操作，或操作后需要人工干预处理

19.3.3日志辅助表结构

高级日志辅助表"表名$ALOG"的结构如下：

序号	列	数据类型	说明

1	ORG_ROWID	BIGINT	源表 ROWID。当 OP_TYPE=0 时，ORG_ROWID=0
2	OP_TYPE	SMALLINT	登记记录日志动作。 0:TRUNCATE 1:行插入 2:批量插入起始 3:批量插入结束 4:更新 5:删除 6:删除后再插入(仅用于堆表)
3	COLMAP	VARBINARY(2048)	当 OP_TYPE=3 时，记录的是批量插入结束的 ROWID; 当 OP_TYPE=4 时，是记录的更新列的列号。例如 0xA3，即二进制的 10100011，表示更新的列为第 1、2、6、8 列，与 DM_BIT_TEST()配合使用; 其他情况为 null
4	COL_0	与源表的第一个主键列类型相同	源表的第一个主键列
5	COL_1	与源表的第二个主键列类型相同	源表的第二个主键列
6	Col_n	...	...

19.3.4系统过程

高级日志辅助表中的 COLMAP 列记录的数据，用&操作只能获取前 64 列的更新情况，因为会数据溢出。

增加系统过程 DM_BIT_TEST（）用于获取一个 VARBINARY 数据的第 N 位的数值。

语法格式

```
DM_BIT_TEST(DATA  varbinary, nth  int);
```

功能：返回二进制数据 varbinary 第 nth 位是 0 还是 1(最低位序号为 1)。如果超过了位数则返回 0。

例 0xF1 转为二进制后为 11110001，从低位开始第 5 位为 1。二进制 1011 从低位开始第三位为 0。

```
SQL> SELECT DM_BIT_TEST(0xF1,5),DM_BIT_TEST(1011,3);
行号       DM_BIT_TEST(0xF1,5) DM_BIT_TEST(1011,3)
---------- ------------------- ------------------
  1           1                    0
```

19.4使用高级日志同步数据的原则

用户根据表定义创建数据同步的目标表，自己编写同步 DMSQL 脚本来进行同步。对于同步，建议遵守如下的原则：

1. 如果源表有主键，如果用户没有特殊的限制或要求，目标表最好也设置同样的主键。
2. 如果源表没有主键，为了准确同步，最好在目标表上添加一个辅助同步的主键列，同步时将 org_rowid 列的值插入该列中。

3． 用户同步数据的脚本基本逻辑如下：

```
declare
   /*遍历日志表的游标*/
   cursor c IS select * from t01$alog for update;
   /*同步用的变量*/
   r t01$alog %rowtype;
   /*同步批量插入用的变量*/
   bi_start t01$alog %rowtype;
   org_rec t01%rowtype;
begin
  /*遍历日志表，根据各记录的 op_type 进行同步*/
  open c;
  loop
     fetch c into r;
     exit when c%notfound;
       if (r.op_type = 0) then
         print 'truncate' ;
         execute immediate 'truncate table t01';
       elseif (r.op_type = 1 or r.op_type = 6) then
         print 'insert ' || r.org_rowid;
         execute immediate 'insert ....'
       elseif (r.op_type = 2) then
         bi_start = r;
         print 'batch insert start';
       elseif (r.op_type = 3) then
         print 'batch insert last ' || bi_start.org_rowid || ' ' ||
cast( r.colmap as bigint);
         execute immediate 'insert ....'
       elseif (r.op_type = 4) then
         print 'update ' || r.org_rowid;
    select * into org_rec from t01 where ……;
         execute immediate 'update ....' using bi_start… r…;
       elseif (r.op_type = 5) then
         print 'delete '  || r.org_rowid;
         execute immediate 'delete ....'
       end if;
  end loop;
  close c;
  /*清理日志表*/
  execute immediate 'alter table t01 truncate advanced log';
end;
/
```

4． 如果在数据同步时源表仍有并发的 DML，脚本中查询日志时要使用 for update 子句。

5． 同步脚本根据源表的结构有所不同：

1) 如果源表有聚集主键

在同步时可使用日志辅助表中的 org_rowid 和主键列辅助源表定位。使用主键列定位目标表。

2） 如果源表有主键，但不是聚集主键

直接根据 org_rowid 定位数据，最好不要使用主键列来定位源表。主键列仅用来定位目标表。

如果该情况下更新了主键列，对于聚集主键，将是删除后更新，如果不是聚集主键，仍是记录更新，日志辅助表中的主键列仍是原值，所以非聚集主键时主键列不要用来定位源表。

3） 如果没有主键

使用 org_rowid 来进行源表的定位；目标表的定义根据用户自己的方式使用 org_rowid 定位。

6． 如果源表中没有聚集索引，批量插入时可以根据 OP_TYPE=3 时的 org_rowid(批量插入起始 ROWID)和 COLMAP 中的数据(批量插入结束 ROWID)范围查询源表插入目标表；如果有聚集索引，考虑到组合索引无法进行范围查询，只能使用第一个主键和 rowid 进行范围查询。

7． MPP 环境下，因为高级日志表是本地表，所以同步数据的时候，只能各个节点单独做同步。

19.5应用实例

19.5.1创建不带主键的源表

1． 创建源表

```
Create table t01(a int, b int, c varchar);
insert into t01 values(88,88, '原始数据 1');
insert into t01 values(99,99, '原始数据 2');
```

2． 在源表上创建日志辅助表

```
Alter table t01  with advanced log;
```

3． 查看日志辅助表结构

```
SQL> Select tabledef('SYSDBA','T01$ALOG');
行号       TABLEDEF('SYSDBA','T01$ALOG')
-------------------------------------------------------------------
1         CREATE TABLE "SYSDBA"."T01$ALOG"
          (
          "ORG_ROWID" BIGINT NOT NULL,
          "OP_TYPE" SMALLINT NOT NULL,
          "COLMAP" VARBINARY(2048),
          CLUSTER PRIMARY KEY("ORG_ROWID", "OP_TYPE")) STORAGE(ON "MAIN",
CLUSTERBTR) ;
```

4． 在源表中删除 1 行数据。

```
delete from t01 where a=88;
SQL> Select * from t01$alog;
```

```
行号        ORG_ROWID            OP_TYPE     COLMAP
---------- -------------------- ----------- ----------
1          1                    5           NULL
```

5. 在源表中更新 1 行数据。

```
update t01 set c='hello world' where a=99;
SQL> Select * from t01$alog;
行号        ORG_ROWID            OP_TYPE     COLMAP
---------- -------------------- ----------- ----------
1          1                    5           NULL
2          2                    4           0x04
```

6. 在源表中再次更新同 1 行数据。这一操作在日志表中没有记录。因为将源表上一条（99,99, '原始数据 2'）的数据更新为（99,99,'hello world'）之后，又再次更新为（99,99,'hello world!'）。这两步更新操作的最终结果就和直接更新为（99,99,'hello world!'）一样，所以两步操作只有一条记录。

```
SQL> update t01 set c='hello world!' where a=99;
行号        ORG_ROWID            OP_TYPE     COLMAP
---------- -------------------- ----------- ----------
1          1                    5           NULL
2          2                    4           0x04
```

7. 先清空源表数据，再查看日志辅助表的变化。发现日志辅助表中也清空了之前的记录，只记录下了清空源表的操作。

```
Truncate table t01;
SQL> Select * from t01$alog;
行号        ORG_ROWID            OP_TYPE     COLMAP
---------- -------------------- ----------- ----------
1          0                    0           NULL
```

8. 在源表中批量插入 100 行数据。单机情况下，大于 100 条才叫批量插入。

```
insert into t01 select level a,level+1 b,level c connect by level<=100 order by a,b;
SQL> Select * from t01$alog;
行号        ORG_ROWID            OP_TYPE     COLMAP
---------- -------------------- ----------- ------------------
1          0                    0           NULL
2          1                    2           NULL
3          1                    3           0x0000000000000064
```

9. 在源表中插入 1 行数据。

```
insert into t01 values(1001,1002,1003);
SQL> Select * from t01$alog;
行号        ORG_ROWID            OP_TYPE     COLMAP
---------- -------------------- ----------- ------------------
1          0                    0           NULL
2          1                    2           NULL
3          1                    3           0x0000000000000064
4          101                  1           NULL
```

10. 同步数据

创建 huge 表。因为不带主键，为了准确同步，在目标表 huge_t01 上添加一个辅助同步的主键列 c_rowid，同步时将 org_rowid 列的值插入该列中

```
create huge table huge_t01 (c_rowid bigint, a int, b int, c varchar(1024));
```

运行同步脚本。同步脚本由用户根据实际情况自行编写。本例中脚本如下：

```
declare
   /*遍历日志表的游标*/
   cursor c IS select * from t01$alog for update;
   /*同步用的变量*/
   r t01$alog %rowtype;
   /*同步批量插入用的变量*/
   bi_start t01$alog %rowtype;
   set_sql    varchar;
   upd_sql    varchar;
   i  int;
begin
  /*遍历日志表，根据各记录的 op_type 进行同步*/
  open c;
  loop
     fetch c into r;
     exit when c%notfound;
       if (r.op_type = 0) then
         print 'truncate' ;
         execute immediate 'truncate table huge_t01;';
       elseif (r.op_type = 1 or r.op_type = 6) then
         print 'insert ' || r.org_rowid;
         execute immediate 'insert into huge_t01 select rowid,* from t01 where
rowid=?;' using r.org_rowid;
       elseif (r.op_type = 2) then
         bi_start = r;
         print 'batch insert start';
       elseif (r.op_type = 3) then
         print 'batch insert last ' || bi_start.org_rowid || ' ' ||
cast( r.colmap as bigint);
         execute immediate 'insert into huge_t01 select rowid,* from t01 where
rowid>= ? and rowid<= ?;' using r.org_rowid, cast(r.colmap as bigint);
       elseif (r.op_type = 4) then
         print 'update ' || r.org_rowid;
         set_sql = '';
         i      = 0;
         if (dm_bit_test(r.colmap,1)) = 1 then set_sql = set_sql || 'a = org.a';
i = i+1; end if;
         if (dm_bit_test(r.colmap,2)) = 1 then  if i > 0 then set_sql = set_sql
||','; end if; set_sql = set_sql || 'b = org.b'; i = i+1; end if;
```

```
          if (dm_bit_test(r.colmap,3)) = 1 then  if i > 0 then set_sql = set_sql ||','; end if; set_sql = set_sql || 'c = org.c'; i = i+1; end if;
          upd_sql = 'declare org t01%rowtype; begin select * into org from t01 where rowid=?; update huge_t01 set ' || set_sql || ' where c_rowid=?; end;';
          execute immediate upd_sql using r.org_rowid, r.org_rowid;
        elseif (r.op_type = 5) then
          print 'delete '  || r.org_rowid;
          execute immediate 'delete from huge_t01 where c_rowid=?;' using r.org_rowid;
        end if;
    end loop;
    close c;
    /*清理日志表*/
    execute immediate 'alter table t01 truncate advanced log';
  end;
  /
```

11. 查询 huge 表中的数据。可以看出，huge_t01 上的数据都是源表创建了日志辅助表之后的增量数据。

```
SQL> Select count(*) from huge_t01;
行号       COUNT(*)
---------- --------------------
1          101
```

19.5.2创建带主键的源表

1. 创建带有日志辅助表的源表

```
Create table t01(a int, b int, c varchar, primary key(a,b)) with advanced log;
```

2. 查看日志辅助表结构

```
SQL> Select tabledef('SYSDBA','T01$ALOG');
行号       TABLEDEF('SYSDBA','T01$ALOG')
---------------------------------------------------------------------------
1          CREATE TABLE "SYSDBA"."T01$ALOG"
           (
           "ORG_ROWID" BIGINT NOT NULL,
           "OP_TYPE" SMALLINT NOT NULL,
           "COLMAP" VARBINARY(2048),
           "COL_0" INTEGER,
           "COL_1" INTEGER,
           CLUSTER PRIMARY KEY("ORG_ROWID", "OP_TYPE")) STORAGE(ON "MAIN", CLUSTERBTR);
```

3. 清空源表

```
Truncate table t01;
SQL> Select * from t01$alog;
行号     ORG_ROWID           OP_TYPE     COLMAP     COL_0       COL_1
```

```
 -------------------- ----------- ---------- ----------- -----------
1         0                  0          NULL       NULL        NULL
```

4. 在源表中插入一条记录

```
insert into t01 values(1001,1002,1003);
SQL> Select * from t01$alog;
行号       ORG_ROWID          OP_TYPE     COLMAP     COL_0 COL_1
-------------------- ----------- ---------- ----------- -----------
1          0                  0          NULL       NULL        NULL
2          1                  1          NULL       1001        1002
```

5. 同步数据

创建 huge 表。

```
create huge table huge_t01 (a int, b int, c varchar(1024), primary key(a,b));
```

运行同步脚本。同步脚本由用户根据实际情况自行编写。本例中脚本如下：

```
declare
   /*遍历日志表的游标*/
   cursor c IS select * from t01$alog for update;
   /*同步用的变量*/
   r t01$alog %rowtype;
   /*同步批量插入用的变量*/
   bi_start t01$alog %rowtype;
   set_sql    varchar;
   upd_sql    varchar;
   i  int;
begin
  /*遍历日志表，根据各记录的 op_type 进行同步*/
  open c;
  loop
     fetch c into r;
     exit when c%notfound;
       if (r.op_type = 0) then
         print 'truncate' ;
         execute immediate 'truncate table huge_t01;';
       elseif (r.op_type = 1 or r.op_type = 6) then
         print 'insert ' || r.org_rowid;
         execute immediate 'insert into huge_t01 select * from t01 where
rowid=?;' using r.org_rowid;
       elseif (r.op_type = 2) then
         bi_start = r;
         print 'batch insert start';
       elseif (r.op_type = 3) then
         print 'batch insert last ' || bi_start.org_rowid || ' ' ||
cast( r.colmap as bigint);
         execute immediate 'insert into huge_t01 select * from t01 where rowid>= ?
and rowid<= ?;' using r.org_rowid, cast(r.colmap as bigint);
```

```
        elseif (r.op_type = 4) then
          print 'update ' || r.org_rowid;
          set_sql = '';
          i      = 0;
          if (dm_bit_test(r.colmap,1)) = 1 then set_sql = set_sql || 'a = org.a';
i = i+1; end if;
          if (dm_bit_test(r.colmap,2)) = 1 then  if i > 0 then set_sql = set_sql
||','; end if; set_sql = set_sql || 'b = org.b'; i = i+1; end if;
          if (dm_bit_test(r.colmap,3)) = 1 then  if i > 0 then set_sql = set_sql
||','; end if; set_sql = set_sql || 'c = org.c'; i = i+1; end if;
          upd_sql = 'declare org t01%rowtype; begin select * into org from t01
where rowid=?; update huge_t01 set ' || set_sql || ' where a = ? and b = ?; end;';
          execute immediate upd_sql using r.org_rowid, r.col_0, r.col_1;
        elseif (r.op_type = 5) then
          print 'delete '  || r.org_rowid;
          execute immediate 'delete from huge_t01 where a= ? and b = ?;' using
r.col_0, r.col_1;
        end if;
    end loop;
    close c;
    /*清理日志表*/
    execute immediate 'alter table t01 truncate advanced log';
  end;
  /
```

6. 查询 huge 表中的数据。可以看出，huge_t01 上的数据都是源表创建了日志辅助表之后的增量数据。

```
SQL> select * from huge_t01;
行号       A           B          C
---------- ----------- ----------- ----
1         1001       1002       1003
```

附录 1 关键字和保留字

以下不带*号的为关键字，带*号的为系统保留字。

DM 还将保留字进一步划分为 SQL 保留字、DMSQL 程序保留字、模式保留字、变量保留字和别名保留字，具体请查询系统视图 V$RESERVED_WORDS。

注意：关键字 ROWID、TRXID、VERSIONS_STARTTIME、VERSIONS_ENDTIME、VERSIONS_STARTTRXID、VERSIONS_ENDTRXID 和 VERSIONS_OPERATION 不能作为表的列名，即使加上双引号也不行。

A

ABORT、* ABSOLUTE、* ABSTRACT、ACCESSED、ACCOUNT、ACROSS、ACTION、* ADD、* ADMIN、ADVANCED、AFTER、AGGREGATE、* ALL、ALLOW_DATETIME、ALLOW_IP、* ALTER、ANALYZE、* AND、* ANY、APR、ARCHIVE、ARCHIVEDIR、ARCHIVELOG、ARCHIVESTYLE、* ARRAY、* ARRAYLEN、* AS、* ASC、ASCII、ASENSITIVE、* ASSIGN、ASYNCHRONOUS、AT、ATTACH、* AUDIT、AUG、AUTHID、* AUTHORIZATION、AUTO、AUTOEXTEND、AUTONOMOUS_TRANSACTION、AVG

B

BACKED、BACKUP、BACKUPDIR、BACKUPINFO、BACKUPSET、BADFILE、BAKFILE、BASE、BEFORE、* BEGIN、* BETWEEN、* BIGDATEDIFF、BIGINT、BINARY、BIT、BITMAP、BLOB、BLOCK、* BOOL、BOOLEAN、* BOTH、BRANCH、BREADTH、* BREAK、* BSTRING、BTREE、BUFFER、BUILD、BULK、* BY、BYDAY、BYHOUR、BYMINUTE、BYMONTH、BYMONTHDAY、BYSECOND、* BYTE、BYWEEKNO、BYYEARDAY

C

CACHE、CALCULATE、* CALL、CASCADE、CASCADED、* CASE、* CAST、CATALOG、* CATCH、CHAIN、* CHAR、CHARACTER、CHARACTERISTICS、* CHECK、CIPHER、* CLASS、CLOB、CLOSE、* CLUSTER、* CLUSTERBTR、COLLATE、* COLLATION、COLLECT、* COLUMN、COLUMNS、* COMMENT、* COMMIT、COMMITTED、* COMMITWORK、COMPILE、COMPLETE、COMPRESS、COMPRESSED、CONDITIONAL、* CONNECT、CONNECT_BY_ISCYCLE、CONNECT_BY_ISLEAF、* CONNECT_BY_ROOT、CONNECT_IDLE_TIME、CONNECT_TIME、* CONST、CONSTANT、* CONSTRAINT、CONSTRAINTS、CONSTRUCTOR、* CONTAINS、* CONTEXT、* CONTINUE、* CONVERT、COPY、* CORRESPONDING、CORRUPT、COUNT、COUNTER、CPU_PER_CALL、CPU_PER_SESSION、* CREATE、* CROSS、* CRYPTO、CTLFILE、* CUBE、CUMULATIVE、* CURRENT、CURRENT_SCHEMA、CURRENT_USER、* CURSOR、CYCLE

D

DAILY、DANGLING、DATA、DATABASE、DATAFILE、DATE、* DATEADD、* DATEDIFF、* DATEPART、DATETIME、DAY、DBFILE、DDL、DDL_CLONE、DEBUG、DEC、* DECIMAL、* DECLARE、* DECODE、* DEFAULT、DEFERRABLE、DEFERRED、DEFINER、* DELETE、DELETING、DELIMITED、DELTA、DEMAND、DENSE_RANK、DEPTH、DEREF、* DESC、DETACH、DETERMINISTIC、DEVICE、DIAGNOSTICS、DICTIONARY、DIRECTORY、* DISABLE、DISCONNECT、* DISKSPACE、* DISTINCT、* DISTRIBUTED、* DO、* DOMAIN、* DOUBLE、DOWN、* DROP、DUMP

E

EACH、* ELSE、* ELSEIF、* ELSIF、EMPTY、* ENABLE、ENCRYPT、ENCRYPTION、* END、* EQU、ERROR、ERRORS、ESCAPE、EVENTINFO、EVENTS、EXCEPT、EXCEPTION、EXCEPTIONS、EXCEPTION_INIT、* EXCHANGE、EXCLUDE、EXCLUDING、EXCLUSIVE、* EXEC、* EXECUTE、* EXISTS、* EXIT、* EXPLAIN、EXTENDS、* EXTERN、EXTERNAL、EXTERNALLY、* EXTRACT

F

FAILED_LOGIN_ATTEMPS、FAST、FEB、* FETCH、FIELDS、FILE、FILEGROUP、FILESIZE、FILLFACTOR、* FINAL、* FINALLY、* FIRST、* FLOAT、FOLLOWING、* FOR、FORALL、FORCE、* FOREIGN、FORMAT、FREQ、FREQUENCE、FRI、* FROM、* FULL、* FULLY、* FUNCTION

G

* GET、GLOBAL、GLOBALLY、* GOTO、* GRANT、* GROUP、* GROUPING

H

HASH、* HAVING、HEXTORAW、HOLD、HOUR、HOURLY、HUGE

I

IDENTIFIED、* IDENTITY、IDENTITY_INSERT、* IF、IMAGE、* IMMEDIATE、* IN、INCLUDE、INCLUDING、INCREASE、INCREMENT、* INDEX、INDEXES、INDICES、INITIAL、INITIALIZED、INITIALLY、* INLINE、* INNER、INNERID、INPUT、INSENSITIVE、* INSERT、INSERTING、INSTANTIABLE、INSTEAD、* INT、INTEGER、INTENT、* INTERSECT、* INTERVAL、* INTO、INVISIBLE、* IS、ISOLATION

J

JAN、* JAVA、JOB、* JOIN、JSON、* JSON_QUERY、* JSON_VALUE、JUL、JUN

K

* KEEP、KEY、KEYS

L

LABEL、LARGE、LAST、LAX、* LEADING、* LEFT、LESS、LEVEL、LEXER、* LIKE、LIMIT、* LINK、* LIST、* LNNVL、LOB、LOCAL、LOCALLY、LOCATION、LOCK、LOCKED、LOG、LOGFILE、LOGGING、LOGIC、* LOGIN、LOGOFF、LOGON、LOGOUT、LONG、LONGVARBINARY、LONGVARCHAR、* LOOP、LSN

M

MANUAL、MAP、MAPPED、MAR、MATCH、MATCHED、MATERIALIZED、MAX、MAXPIECESIZE、MAXSIZE、MAXVALUE、MAX_RUN_DURATION、MAY、* MEMBER、MEMORY、MEM_SPACE、MERGE、MIN、MINEXTENTS、* MINUS、MINUTE、MINUTELY、MINVALUE、MIRROR、MOD、MODE、MODIFY、MON、MONEY、MONITORING、MONTH、MONTHLY、MOUNT、MOVEMENT、* MULTISET

N

NATIONAL、* NATURAL、NCHAR、NCHARACTER、NEVER、* NEW、* NEXT、NO、NOARCHIVELOG、NOAUDIT、NOBRANCH、NOCACHE、* NOCOPY、* NOCYCLE、NODE、NOLOGGING、NOMAXVALUE、NOMINVALUE、NOMONITORING、NONE、NOORDER、NORMAL、NOROWDEPENDENCIES、NOSORT、* NOT、NOT_ALLOW_DATETIME、NOT_ALLOW_IP、NOV、NOWAIT、* NULL、NULLS、NUMBER、NUMERIC

O

* OBJECT、OCT、* OF、OFF、OFFLINE、OFFSET、OLD、* ON、ONCE、ONLINE、ONLY、OPEN、OPTIMIZE、OPTION、* OR、* ORDER、* OUT、OUTER、* OVER、OVERLAPS、* OVERLAY、

* OVERRIDE、OVERRIDING

P

PACKAGE、PAD、PAGE、PARALLEL、PARALLEL_ENABLE、PARMS、PARTIAL、* PARTITION、PARTITIONS、PASSING、PASSWORD_GRACE_TIME、PASSWORD_LIFE_TIME、PASSWORD_LOCK_TIME 、 PASSWORD_POLICY 、 PASSWORD_REUSE_MAX 、 PASSWORD_REUSE_TIME、PATH、* PENDANT、* PERCENT、PIPE、PIPELINED、PIVOT、PLACING、PLS_INTEGER、PRAGMA、PRECEDING、PRECISION、PRESERVE、PRETTY、* PRIMARY、* PRINT、* PRIOR、* PRIVATE、PRIVILEGE、* PRIVILEGES、* PROCEDURE、* PROTECTED、* PUBLIC、PURGE

Q

QUERY_REWRITE_INTEGRITY

R

* RAISE、RANDOMLY、RANGE、RAWTOHEX、READ、READONLY、READ_PER_CALL、READ_PER_SESSION、REAL、REBUILD、* RECORD、RECORDS、* REF、* REFERENCE、* REFERENCES、* REFERENCING、REFRESH、RELATED、* RELATIVE、RENAME、* REPEAT、REPEATABLE、REPLACE、REPLAY、* REPLICATE、RESIZE、RESTORE、RESTRICT、RESULT、RESULT_CACHE、* RETURN、* RETURNING、* REVERSE、* REVOKE、* RIGHT、ROLE、* ROLLBACK、ROLLFILE、* ROLLUP、ROOT、* ROW、ROWCOUNT、ROWDEPENDENCIES、ROWID、* ROWNUM、* ROWS、RULE

S

SALT、SAMPLE、SAT、SAVE、* SAVEPOINT、* SBYTE、* SCHEMA、SCOPE、SCROLL、* SEALED、SEARCH、SECOND、SECONDLY、* SECTION、SEED、* SELECT、SELF、SENSITIVE、SEP、SEQUENCE、SERERR、SERIALIZABLE、SERVER、SESSION、SESSION_PER_USER、* SET、* SETS、SHARE、* SHORT、SHUTDOWN、SIBLINGS、SIMPLE、SINCE、SIZE、* SIZEOF、SKIP、SMALLINT、SNAPSHOT、* SOME、SOUND、SPACE、SPAN、SPATIAL、SPFILE、SPLIT、SQL、STANDBY、STARTUP、STAT、STATEMENT、* STATIC、STDDEV、STORAGE、STORE、STRICT、STRING、* STRUCT、STYLE、* SUBPARTITION、SUBPARTITIONS、SUBSTRING、SUBTYPE、SUCCESSFUL、SUM、SUN、SUSPEND、* SWITCH、SYNC、SYNCHRONOUS、* SYNONYM、SYSTEM、SYS_CONNECT_BY_PATH

T

* TABLE、TABLESPACE、TASK、TEMPLATE、TEMPORARY、TEXT、THAN、THEN、THREAD、* THROW、THU、TIES、TIME、TIMER、TIMES、TIMESTAMP、* TIMESTAMPADD、* TIMESTAMPDIFF、TIME_ZONE、TINYINT、* TO、* TOP、TRACE、* TRAILING、TRANSACTION、TRANSACTIONAL、* TRIGGER、TRIGGERS、* TRIM、* TRUNCATE、TRUNCSIZE、TRXID、* TRY、TUE、TYPE、* TYPEDEF、* TYPEOF

U

* UINT、* ULONG、UNBOUNDED、UNCOMMITTED、UNCONDITIONAL、UNDER、* UNION、* UNIQUE、UNLIMITED、UNLOCK、UNPIVOT、* UNTIL、UNUSABLE、UP、* UPDATE、UPDATING、USAGE、* USER、USE_HASH、USE_MERGE、USE_NL、USE_NL_WITH_INDEX、* USHORT、* USING、VALUE、* VALUES、VARBINARY、VARCHAR、VARCHAR2、VARIANCE

V

* VARRAY 、 VARYING 、 * VERIFY 、 VERSIONS 、 VERSIONS_ENDTIME 、 VERSIONS_ENDTRXID 、 VERSIONS_OPERATION 、 VERSIONS_STARTTIME 、 VERSIONS_STARTTRXID、VERTICAL、* VIEW、* VIRTUAL、* VISIBLE、* VOID、VSIZE

W

WAIT、WED、WEEK、WEEKLY、* WHEN、* WHENEVER、* WHERE、* WHILE、* WITH、*WITHIN、WITHOUT、WORK、WRAPPED、WRAPPER、WRITE

X

XML、* XMLPARSE、* XMLTABLE

Y

YEAR、YEARLY

Z

ZONE

附录 2 SQL 语法描述说明

< >　　表示一个语法对象。

::=　　定义符，用来定义一个语法对象。定义符左边为语法对象，右边为相应的语法描述。

|　　或者符，或者符限定的语法选项在实际的语句中只能出现一个。

{ }　　大括号指明大括号内的语法选项在实际的语句中可以出现 0…N 次（N 为大于 0 的自然数），但是大括号本身不能出现在语句中。

[ ]　　中括号指明中括号内的语法选项在实际的语句中可以出现 0…1 次，但是中括号本身不能出现在语句中。

关键字　关键字在 DM_SQL 语言中具有特殊意义，在 SQL 语法描述中，关键字以大写形式出现。但在实际书写 SQL 语句时，关键字可以为大写也可以为小写。

SQL 语法图的说明

SQL 语法图是用来帮助用户正确地理解和使用 DM SQL 语法的图形。阅读语法图时，请按照从上到下，从左到右的方式，依箭头所指方向进行阅读。

SQL 命令、语法关键字等终结符以全大写方式在长方形框内显示，使用时直接输入这些内容；语法参数或语法子句等非终结符的名称以全小写方式在圆角框内显示；各类标点符号显示在圆圈之中。注：如果小写参数中不带下划线_，则表示是由用户输入的参数，带下划线则表示是还需要进一步解释的子句或语法对象，如果在前面已解释过，则未重复列出。

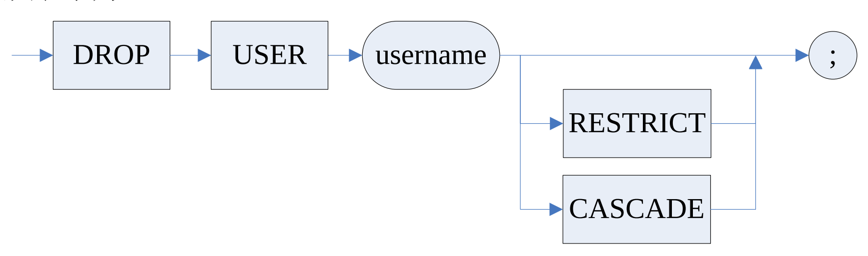
1.必须关键字和参数

必须关键字和参数出现在语法参考图的主干路径上，也就是说，出现在当前阅读的水平线上。例如：

用户删除语句

```
DROP USER <用户名> [RESTRICT | CASCADE];
```

用语法图表示为



这里 DROP、USER、username 和;都是语句必须的。

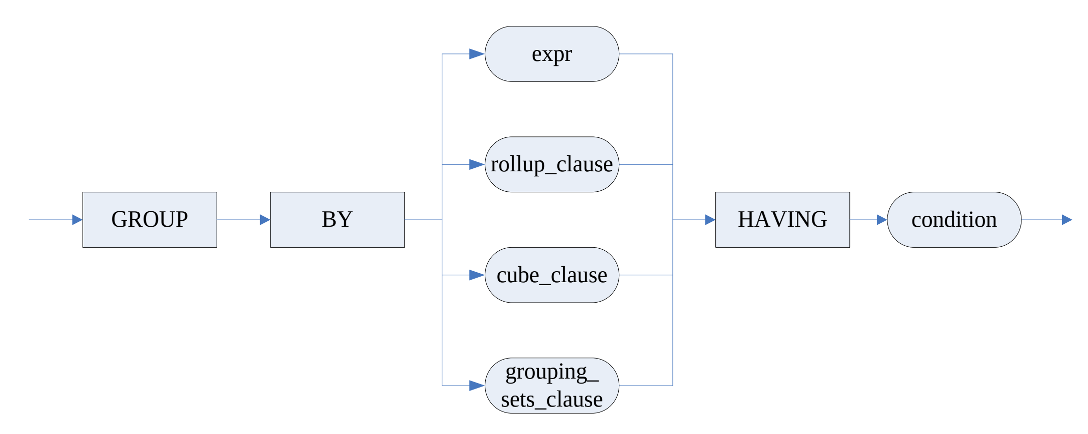
如果多个关键字或参数并行地出现在从主路径延伸出的多条分支路径中，则他们之中有一个是必须的。也就是说，必须选择他们其中的一个，但不需要一定是主路径上的那个。

例如：

```
<GROUP BY 子句> ::= GROUP BY <分组项> | <ROLLUP 项> | <CUBE 项> | <GROUPING SETS 项>
```

用语法图表示

GROUP BY 子句



从语法图可以看出，GROUP BY 子句有四种形式，可以任选一种。如果存在 GROUP BY 子句，则必须选择其中一种。

2．可选关键字和参数

如果关键字或参数并行地出现在主路径下方，而主路径是一条直线，则这些关键字和参数是可选的。例如：

```
DROP USER <用户名> [RESTRICT | CASCADE];
```

用语法图表示为：

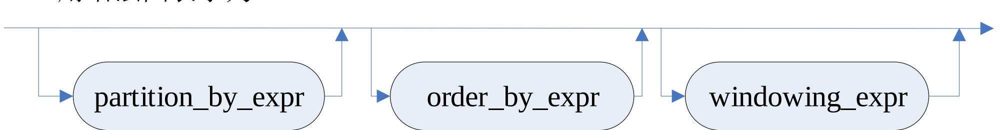
DROP
USER
username
;
RESTRICT
CASCADE

从语法图可以看出，DROP、USER、username 是必须的。而 RESTRICT 和 CASCADE 都是可选的，也可以选择直接通过主路径，而不需走这些并行路径。这些并行路径能且只能选择其中一种。又如：

分析函数的分析子句语法：

```
<分析子句> ::= [<PARTITION BY 项>] [<ORDER BY 项> ] [<窗口子句>]
```

用语法图表示为



其中，partition_by_expr、order_by_expr 和 windowing_expr 参数都是可选的，但如果出现，则出现的顺序不能颠倒。

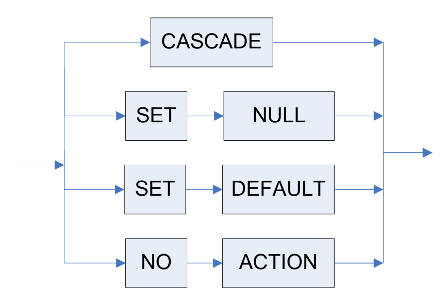
3．多条路径

如果一张语法参考图有一条以上的路径，可以从任意一条路径进行阅读。如果可以选择多个关键字、操作符、参数或者语法子句，这些选项将被并行地列出。例如：

```
<引用动作> ::= [CASCADE] | [SET NULL] | [SET DEFAULT] | [NO ACTION]
```

用语法图表示为

ref_action



从语法图可以看出，引用动作可以选择这四种的任一种。

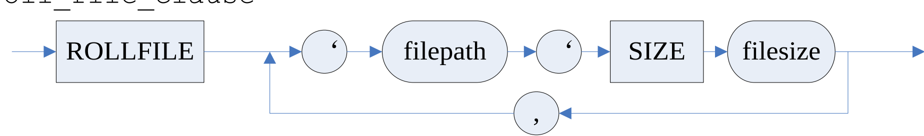
4.循环语法

循环语法表示可以按照需要，使用循环内的语法一次或者多次。例如：

```
<回滚文件子句> ::= ROLLFILE <文件说明子句>,{<文件说明子句>}
<文件说明子句> ::= <文件路径> SIZE <文件大小>
```

用语法图表示为

roll_file_clause



从语法图可以看出，通过逗号隔开，可以重复语法对象 ‘filepath’ SIZE filesize 多个。

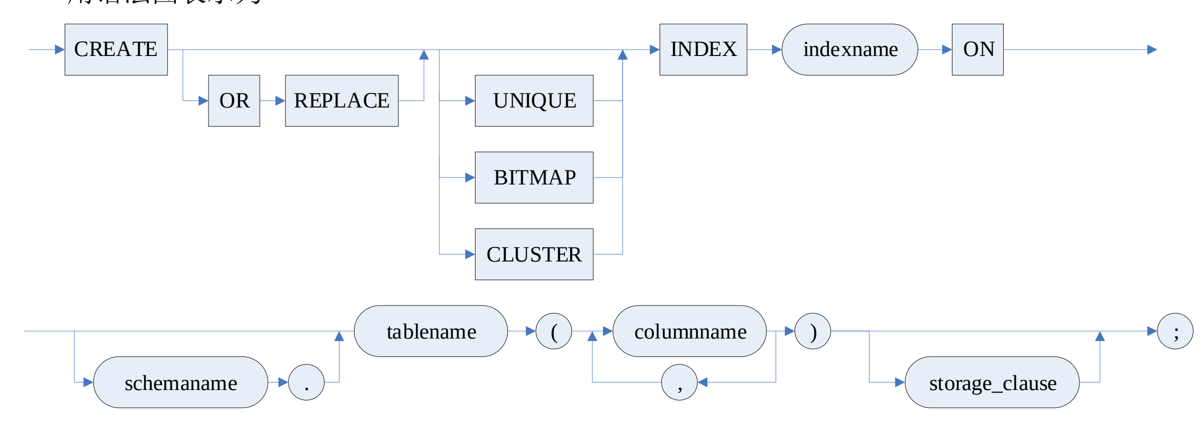
5.多行语法图

由于有些 SQL 语句的语法十分复杂，生成的语法参考图无法完整地显示在一行之内，于是他们将被分行显示。阅读此类图形时，请从上至下，从左至右地进行。例如：

索引定义语句

```
CREATE [OR REPLACE] [UNIQUE | BITMAP | CLUSTER] INDEX <索引名>
ON [<模式名>.]<表名>(<列名>{,<列名>}) [<STORAGE 子句>];
```

用语法图表示为



从语法图可以看出，索引定义的语法被分成了两行显示。

附录 3 系统存储过程和函数

以下为达梦数据库所用到的系统存储过程和函数。

注：函数名右上角有“*”标记的，表示此过程/函数的使用有下面两点限制：

1. 此过程/函数不能在 MPP 全局模式下的存储过程中直接调用，在 MPP LOCAL 模式下可在存储过程中直接调用；
2. 此过程/函数不能在存储过程中带参数进行动态调用。

1) INI 参数管理

1) SF_GET_PARA_VALUE

定义：

```
int
SF_GET_PARA_VALUE (
  scope          int,
  ini_param_name  varchar(256)
)
```

功能说明：

返回 dm.ini 文件中非浮点和字符串类型的参数值

参数说明：

scope： 取值为 1、2 。 1 表示从 dm.ini 文件中读取；2 表示从内存中读取。当取值为 1，且 dm.ini 文件中该参数值设置为非法值时，若设置值与参数类型不兼容，则返回默认值；若设置值小于参数取值范围的最小值，则返回最小值；若设置值大于参数取值范围的最大值，则返回最大值

ini_param_name：dm.ini 文件中的参数名

返回值：

当前 INI 文件中对应的参数值

举例说明：

获得 dm.ini 文件中 BUFFER 参数值

```
SELECT SF_GET_PARA_VALUE (1, 'BUFFER');
```

2) SP_SET_PARA_VALUE*

定义：

```
void
SP_SET_PARA_VALUE (
  scope          int,
  ini_param_name  varchar(256)
  value          bigint
)
```

功能说明：

设置 dm.ini 文件中非浮点和字符串类型的参数值

参数说明：

scope：取值为 1，2 。1 表示 dm.ini 文件和内存参数都修改，不需要重启服务器；2 表示只修改 dm.ini 文件，服务器重启后生效。

ini_param_name：dm.ini 文件中的参数名。

value：设置的值。

返回值：

无

举例说明：

将 dm.ini 文件中 HFS_CACHE_SIZE 参数值设置为 320

```
SP_SET_PARA_VALUE (1,'HFS_CACHE_SIZE',320);
```

3) SF_GET_PARA_DOUBLE_VALUE

定义：

```
double
SF_GET_PARA_DOUBLE_VALUE (
  scope          int,
  ini_param_name  varchar(256)
)
```

功能说明：

返回dm.ini文件中参数中浮点型的参数值

参数说明：

scope：取值为1，2 。1表示从dm.ini文件中读取；2表示从内存中读取；

ini_param_name：dm.ini文件中的参数名。

返回值：

当前INI文件中对应的参数值

举例说明：

获得dm.ini中SEL_RATE_EQU参数值

```
SELECT SF_GET_PARA_DOUBLE_VALUE (1, 'CKPT_FLUSH_RATE');
SELECT SF_GET_PARA_DOUBLE_VALUE (2, 'CKPT_FLUSH_RATE');
```

4) SP_SET_PARA_DOUBLE_VALUE*

定义：

```
void
SP_SET_PARA_DOUBLE_VALUE (
    scope          int,
    ini_param_name  varchar(256),
    value          double
)
```

功能说明：

设置dm.ini参数中浮点型的参数值

参数说明：

scope：取值为 1，2。1 表示 dm.ini 文件和内存参数都修改，不需要重启服务器；2 表示只可修改 dm.ini 文件，服务器重启后生效。

ini_param_name：dm.ini文件中的参数名。

value：设置的值。

返回值：

无

举例说明：

将dm.ini文件中SEL_RATE_EQU参数值设置为0.3

```
SP_SET_PARA_DOUBLE_VALUE(1, 'SEL_RATE_EQU', 0.3);
```

5) SF_GET_PARA_STRING_VALUE

定义：

```
char*
SF_GET_PARA_STRING_VALUE (
    scope          int,
    ini_param_name  varchar(256)
)
```

功能说明：

返回 dm.ini 文件中字符串类型的参数值

参数说明：

scope： 取值为1，2 。 1表示从dm.ini文件中读取；2表示从内存中读取；

ini_param_name：dm.ini文件中的参数名

返回值：

当前 INI 文件中对应的参数值

举例说明：

获得dm.ini文件中TEMP_PATH参数值

```
SELECT SF_GET_PARA_STRING_VALUE (1, 'TEMP_PATH');
```

6) SP_SET_PARA_STRING_VALUE*

定义：

```
void
SP_SET_PARA_STRING_VALUE (
  scope          int,
  ini_param_name  varchar(256) ,
  value          varchar(8187)
)
```

功能说明：

设置 dm.ini 文件中的字符串型参数值

参数说明：

scope：取值为 1，2 。1 表示 dm.ini 文件和内存参数都修改，不需要重启服务器；2 表示只修改 dm.ini 文件，服务器重启后生效。

ini_param_name：dm.ini 文件中的参数名。

value：设置的字符串的值。

返回值：

无

举例说明：

将 dm.ini 文件中 SQL_TRACE_MASK 参数值设置为 1

```
SP_SET_PARA_STRING_VALUE(1, 'SQL_TRACE_MASK','1');
```

7) SF_SET_SESSION_PARA_VALUE*

定义：

```
void
SF_SET_SESSION_PARA_VALUE (
    paraname   varchar(8187),
    value      bigint
)
```

功能说明：

设置会话级 INI 参数的值

参数说明：

paraname： 会话级 INI 参数的参数名

value：要设置的新值

返回值：

无

举例说明：

设置会话级 ini 参数 JOIN_HASH_SIZE 的值为 2000

```
SF_SET_SESSION_PARA_VALUE ('JOIN_HASH_SIZE', 2000);
```

8) SP_RESET_SESSION_PARA_VALUE*

定义：

```
void
SP_RESET_SESSION_PARA_VALUE (
    paraname   varchar(8187)   )
```

功能说明：

重置会话级 INI 参数的值，使得参数的值和系统级一致。

参数说明：

paraname：会话级 INI 参数的参数名。

返回值：

无

举例说明：

重置会话级 INI 参数 JOIN_HASH_SIZE 的值

```
SP_RESET_SESSION_PARA_VALUE ('JOIN_HASH_SIZE');
```

9) SF_GET_SESSION_PARA_VALUE

定义：

```
int
```

```
SF_GET_SESSION_PARA_VALUE (
    paraname   varchar(8187)   )
```

功能说明：

获得整数类型的会话级 INI 参数的值。

参数说明：

paraname：会话级 INI 参数的参数名。

返回值：

整数类型的会话级 INI 参数的值

举例说明：

获取会话级 INI 参数 JOIN_HASH_SIZE 的值：

```
SELECT SF_GET_SESSION_PARA_VALUE ('JOIN_HASH_SIZE');
```

10) SF_GET_SESSION_PARA_DOUBLE_VALUE

定义：

```
double
SF_GET_SESSION_PARA_DOUBLE_VALUE (
    paraname   varchar(8187)   )
```

功能说明：

获得浮点型的会话级 INI 参数的值。

参数说明：

paraname：会话级 INI 参数的参数名。

返回值：

浮点型会话级 INI 参数的值

举例说明：

获取会话级 INI 参数 SEL_RATE_SINGLE 的值：

```
SELECT SF_GET_SESSION_PARA_DOUBLE_VALUE ('SEL_RATE_SINGLE');
```

11) SF_SET_SYSTEM_PARA_VALUE*

定义：

```
void
SF_SET_SYSTEM_PARA_VALUE (
    paraname varchar(256),
    value bigint\double\varchar(256),
    deferred int,
    scope int)
```

功能说明：

修改系统整型、double、varchar 的静态配置参数或动态配置参数。

参数说明：

paraname：ini 参数的参数名。

value：要设置的新值。

deferred：是否立即生效。为 0 表示当前 session 修改的参数立即生效，为 1 表示当前 session 不生效，后续再生效，默认为 0。

scope：取值为 0、1、2 。0 表示修改内存中的参数值，1 表示修改内存和 INI 文件中参数值，0 和 1 都只能修改动态的配置参数。2 表示修改 INI 文件中参数，此时可用来修改静态配置参数和动态配置参数。

返回值：

无

举例说明：

修改 INI 参数 JOIN_HASH_SIZE 的值：

```
SF_SET_SYSTEM_PARA_VALUE ('JOIN_HASH_SIZE',50,1,1);
```

12) SF_SET_SQL_LOG

定义：

```
INT
SF_SET_SQL_LOG (
    svrlog    int,
    svrmsk    varchar(1000)
)
```

功能说明：

设置服务器日志相关 INI 参数 SVR_LOG 和 SQL_TRACE_MASK

参数说明：

svrlog：INI 参数 SVR_LOG 的设置值

svrmsk：INI 参数 SQL_TRACE_MASK 的设置值

返回值：

是否成功

举例说明：

设置服务器日志相关 INI 参数

```
SELECT  SF_SET_SQL_LOG(1, '3:5:7');
```

2）系统信息管理

1) SP_SET_SESSION_READONLY

定义：

```
INT SP_SET_SESSION_READONLY (
READONLY INT)
```

功能说明：

设置当前会话的只读模式

参数说明：

取值 1 或 0。1 表示对数据库只读，0 表示对数据库为读写

返回值：

无。

举例说明：

设置当前会话为只读模式

```
SP_SET_SESSION_READONLY (1);
```

2) SP_CLOSE_SESSION

定义：

VOID SP_CLOSE_SESSION (

SESSION_ID INT)

功能说明：

停止一个活动的会话，会话 ID 可通过 V$SESSIONS 查询，DM 系统创建的内部 SESSION（PORT_TYPE = 12，但 CONNECTED = 1 的 SESSION）也可通过 V$SESSIONS 查询到。

参数说明：

SESSION_ID：会话 ID

举例说明：

```
SP_CLOSE_SESSION (510180488);
```

3) SF_GET_CASE_SENSITIVE_FLAG/ CASE_SENSITIVE

定义：

INT

SF_GET_CASE_SENSITIVE_FLAG()

或者

INT CASE_SENSITIVE ()

功能说明：

返回大小写敏感信息

参数说明：

无

返回值：

1：敏感

0：不敏感

举例说明：

获得大小写敏感信息

```
SELECT SF_GET_CASE_SENSITIVE_FLAG();
```

4) SF_GET_EXTENT_SIZE

定义：

INT

SF_GET_EXTENT_SIZE()

功能说明：

返回簇大小

参数说明：

无

返回值：

系统建库时指定的簇大小

举例说明：

获得系统建库时指定的簇大小

```
SELECT SF_GET_EXTENT_SIZE ();
```

5) SF_GET_PAGE_SIZE/PAGE

定义：

```
INT
SF_GET_PAGE_SIZE()
```

功能说明：

返回页大小

参数说明：

无

返回值：

系统建库时指定的页大小

举例说明：

获得系统建库时指定的页大小

```
SELECT SF_GET_PAGE_SIZE ();
```

补充说明：

获得系统建库时指定的页大小也可使用：

```
SELECT PAGE();
```

6) SF_PAGE_GET_SEGID

定义：

```
INTEGER
SF_PAGE_GET_SEGID(
    ts_id int,
    file_id int,
    page_no int
)
```

功能说明：

获取目标页所在的段号

参数说明：

ts_id：指定目标页的表空间，如果表空间不存在，报错

file_id：指定目标页的文件，如果文件不存在，报错

page_no：指定目标页的编号，如果编号超出文件范围，报错

返回值：

页所在的段号。如果返回 0，表示该页是描述页，不属于任何段。

举例说明：

```
SELECT SF_PAGE_GET_SEGID(4, 0, 2000);
```

7) SF_PAGE_GET_PAGE_TYPE

定义：

```
VARCHAR
SF_PAGE_GET_PAGE_TYPE(
    ts_id int,
    file_id int,
    page_no int
)
```

功能说明：

获取目标页的类型

参数说明：

ts_id：指定目标页的表空间，如果表空间不存在，报错

file_id：指定目标页的文件，如果文件不存在，报错

page_no：指定目标页的编号，如果编号超出文件范围，报错

返回值：

页的类型，没有使用的页，返回"PAGE_NOUSE"，保留页，返回"RESERVED"

举例说明：

```
SELECT SF_PAGE_GET_PAGE_TYPE(4, 0, 2000);
```

8) SF_GET_FILE_BYTES_SIZE

定义：

```
int
SF_GET_FILE_BYTES_SIZE (
GROUPID         int,
FILEID        int
)
```

功能说明：

获取文件字节长度

参数说明：

GROUPID：所属的表空间 ID

FILEID：数据库文件 ID

返回值：

文件字节长度

举例说明：

获取 0 号文件组中 0 号文件的字节长度

```
SELECT SF_GET_FILE_BYTES_SIZE (0,0);
```

9) SF_GET_UNICODE_FLAG/UNICODE

定义：

```
INT
SF_GET_UNICODE_FLAG()
```

或者

```
INT UNICODE ()
```

功能说明：

返回建库时指定的字符集

参数说明：

无

返回值：

0 表示 GB18030，1 表示 UTF-8，2 表示 EUC-KR

举例说明：

获得系统建库时指定字符集

```
SELECT SF_GET_UNICODE_FLAG ();
```

10) SF_GET_SGUID

定义：

INT

SF_GET_SGUID ()

功能说明：

返回数据库唯一标志 sguid

参数说明：

无

返回值：

返回数据库唯一标志 sguid

举例说明：

获取数据库唯一标志 sguid

```
SELECT SF_GET_SGUID();
```

11) GUID()

定义：

VARCHAR

GUID ()

功能说明：

生成一个唯一编码串（32 个字符）

返回值：

返回一个唯一编码串

举例说明：

获取一个唯一编码串。

```
SELECT GUID();
```

12) NEWID()

定义：

VARCHAR

NEWID ()

功能说明：

生成一个 SQLSERVER 格式的 guid 字符串 SQLSERVER 的 guid 格式 xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

返回值：

返回一个 SQLSERVER 格式的唯一编码串

举例说明：

获取一个唯一编码串。

```
SELECT NEWID();
```

13) SESSID()

定义：

INT

SESSID ( )

功能说明：

获取当前连接的 id

返回值：

返回当前连接 id

举例说明：

返回当前连接 id

```
SELECT  SESSID ();
```

14) CHECK_INDEX

定义：

```
INT CHECK_INDEX (
    schname    varchar,
    indexid  int
)
```

功能说明：

检查某一索引的合法性（正确性和有效性）。检查过程中，会 S+IX 封锁对应的表对象，如果封锁失败，会忽略对应索引检查，并记录日志到 dmserver 的 log 文件中。

参数说明：

schname：模式名

indexid：索引 id

返回值：

0：表示不合法；1：表示合法；2：表示存在未校验的索引。

举例说明：

```
CREATE INDEX PRODUCT_IND ON PRODUCTION.PRODUCT(PRODUCTID);
--查询系统表得到索引 ID
select name, id from sysobjects where name='PRODUCT_IND' and subtype$='INDEX';
select CHECK_INDEX ('PRODUCTION',33555531);
```

15) CHECK_DB_INDEX

定义：

```
INT CHECK_DB_INDEX ()
```

功能说明：

检查数据库中所有索引的合法性（正确性和有效性）。检查过程中，会 S+IX 封锁对应的表对象，如果封锁失败，会忽略对应索引检查，并记录日志到 dmserver 的 log 文件中。

返回值：

0：表示不合法；1：表示合法；2：表示存在未校验的索引。

举例说明：

```
SELECT CHECK_DB_INDEX ();
```

16) UID

定义：

INT

UID ()

功能说明：

返回当前用户 ID

参数说明：

无

返回值：

返回当前用户 ID

举例说明：

返回当前用户 ID

```
SELECT UID();
```

17) USER

定义：

CHAR*

USER ()

功能说明：

返回当前用户名

参数说明：

无

返回值：

返回当前用户名

举例说明：

返回当前用户名

```
SELECT USER();
```

18) CUR_DATABASE

定义：

CHAR*

CUR_DATABASE ()

功能说明：

返回数据库名

参数说明：

无

返回值：

返回数据库名

举例说明：

获取数据库名

```
SELECT CUR_DATABASE();
```

19) VSIZE

定义：

INT

VSIZE(n)

功能说明：

返回 n 的核心内部表示的字节数。如果 n 为 NULL，则返回 NULL。

参数说明：

n：待求字节数的参数，可以为任意数据类型

返回值：

n 占用的字节数

举例说明：

```
SELECT VSIZE(256);
```

查询结果：4　　/*整数类型*/

```
SELECT VSIZE('数据库');
```

查询结果：6　　/*中文字符*/

20) SP_RECLAIM_TS_FREE_EXTENTS

定义：

```
SP_RECLAIM_TS_FREE_EXTENTS (
    tsname        varchar(128)
)
```

功能说明：

重组表空间空闲簇

参数说明：

tsname：表空间名

返回值：

无

举例说明：

重组表空间空闲簇

```
CALL SP_RECLAIM_TS_FREE_EXTENTS('SYSTEM');
```

21) SP_CLEAR_PLAN_CACHE

定义：

```
SP_CLEAR_PLAN_CACHE()
```

功能说明：

清空执行缓存信息。

参数说明：

无

返回值：

无

举例说明：

清空执行缓存信息

```
CALL SP_CLEAR_PLAN_CACHE();
```

22) SP_SET_PLN_RS_CACHE

定义：

```
SP_SET_PLN_RS_CACHE(
  plan_id     bigint,
  to_cache int
```

)

功能说明：

强制设置指定计划结果集缓存的生效及失效

参数说明：

plan_id：指定计划 ID，其值可以从动态视图 V$CACHEPLN 中的 CACHE_ITEM 列获得；

to_cache：指定缓存与否，0：不缓存；1：缓存

返回值：

无

举例说明：

设置计划 ID 为 473546872 的计划结果集缓存生效。

```
CALL SP_SET_PLN_RS_CACHE(473546872, 1);
```

23) SP_CLEAR_PLAN_CACHE

定义：

```
SP_CLEAR_PLAN_CACHE(
  plan_id     bigint
)
```

功能说明：

清空指定的执行缓存信息。

参数说明：

plan_id：指定计划 ID，其值可以从动态视图 V$CACHEPLN 中的 CACHE_ITEM 列获得

返回值：

无

举例说明：

清空 ID 为 473546872 的执行缓存信息

```
CALL SP_CLEAR_PLAN_CACHE(473546872);
```

24) SF_CHECK_USER_TABLE_PRIV

定义：

```
int
SF_CHECK_USER_TABLE_PRIV(
  schema_name  varchar(128),
  table_name   varchar(128),
  user_name    varchar(128),
  priv_code    int
)
```

功能说明：

返回用户对表是否具有某种权限

参数说明：

schema_name： 模式名；

table_name：表名；

user_name：用户名；

priv_code：权限代码，0=SELECT，1=INSERT，2=DELETE，3=UPDATE，4=REFERENCE

返回值：

0：用户不具备相应权限；1：用户具备相应权限

举例说明：

获得用户 SYSDBA 对表 SYS.SYSOBJECTS 的查询权限

```
SELECT SF_CHECK_USER_TABLE_PRIV ('SYS', 'SYSOBJECTS', 'SYSDBA', 0);
```

25) SF_CHECK_USER_TABLE_COL_PRIV

定义：

```
int
SF_CHECK_USER_TABLE_COL_PRIV(
  schema_name  varchar(128),
  table_name   varchar(128),
  col_name     varchar(128),
  user_name    varchar(128),
  priv_code     int
)
```

功能说明：

返回用户对表中某列是否具有某种权限

参数说明：

schema_name：模式名；

table_name：表名；

col_name：列名；

user_name：用户名；

priv_code：权限代码，0=SELECT，1=INSERT，2=DELETE，3=UPDATE，4=REFERENCE

返回值：

0：用户不具备相应权限；1：用户具备相应权限

举例说明：

获得用户 SYSDBA 对表 SYS.SYSOBJECTS 的 ID 列的查询权限

```
SELECT SF_CHECK_USER_TABLE_COL_PRIV ('SYS', 'SYSOBJECTS', 'ID' , 'SYSDBA', 0);
```

26) CUR_TICK_TIME

定义：

```
varchar
CUR_TICK_TIME ()
```

功能说明：

获取系统当前时钟记数

参数说明：

无

返回值：

时钟记数的字符串

举例说明：

获取系统当前时钟记数。

```
SELECT CUR_TICK_TIME();
```

27) SP_SET_LONG_TIME

定义：

```
SP_SET_LONG_TIME (
    long_exec_time   int
)
```

功能说明：

设置 V$LOG_EXEC_SQLS_TIME 动态视图中监控 SQL 语句的最短执行时间，以毫秒为单位，有效范围 50~3600000。仅 INI 参数 ENABLE_MONITOR 值大于 1 时设置有效。

参数说明：

无

返回值：

无

举例说明：

监控执行时间超过 5 秒的 SQL 语句。

```
CALL SP_SET_LONG_TIME(5000);
```

28) SF_GET_LONG_TIME

定义：

```
int
SF_GET_LONG_TIME ()
```

功能说明：

返回 V$LONG_EXEC_SQLS 动态视图中监控的最短执行时间，以毫秒为单位。

参数说明：

无

返回值：

V$LONG_EXEC_SQLS所监控的最短执行时间。

举例说明：

查看 V$LONG_EXEC_SQLS 监控的最短执行时间。

```
SELECT SF_GET_LONG_TIME();
```

29) PERMANENT_MAGIC

定义：

```
INT
PERMANENT_MAGIC ()
```

功能说明：

返回数据库永久魔数

参数说明：

无

返回值：

返回整型值：永久魔数

举例说明：

获取数据库永久魔数

```
SELECT PERMANENT_MAGIC();
```

30) SP_CANCEL_SESSION_OPERATION

定义：

```
SP_CANCEL_SESSION_OPERATION (
        session_id    bigint
)
```

功能说明：

终止指定会话的操作

参数说明：

session_id：指定会话的 id

返回值：

无

举例说明：

终止 ID 为 310509680 的会话的操作

```
CALL SP_CANCEL_SESSION_OPERATION (310509680);
```

31) SP_TRUNC_TS_FILE

定义：

```
SP_TRUNC_TS_FILE (
        ts_id          int,
        file_id        int,
    to_size       int
)
```

功能说明：

将临时表空间文件截断到指定的大小，仅能截断文件空闲未使用的部分，如果在指定 to_size 后存在已使用数据页，则实际截断大小会大于 to_size。

参数说明：

ts_id：指定截断文件的临时表空间 ID

file_id：指定截断文件的文件 ID

to_size：指定将文件截断至多大，以 M 为单位；to_size 大小换算成页数后，值必须在 4096 到 2G 之间

返回值：

无

举例说明：

将临时表空间文件号为 0 的文件截断到 32M 大小，库的页大小为 8K。

```
CALL SP_TRUNC_TS_FILE (3, 0, 32);
```

32) SF_GET_SESSION_SQL

定义：

CLOB

```
SF_GET_SESSION_SQL (
        sess_id           bigint
)
```

功能说明：

返回指定会话上最近处理的完整的 SQL 语句。

参数说明：

sess_id：指定会话的会话 ID，对应 V$SESSIONS 视图的 SESS_ID 字段。

返回值：

指定会话上最近处理的完整的 SQL 语句

举例说明：

在 ID 为 96710784 的会话上执行如下语句：

```
CREATE OR REPLACE PROCEDURE xx  AS
BEGIN
SELECT SF_GET_SESSION_SQL(96710784);
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
/
```

重新打开一个会话，调用 xx 过程。可以查看到 ID 为 96710784 的会话上的最后一次执行的 SQL 语句。

```
SQL> call xx;
```

执行结果为：

```
行号      SF_GET_SESSION_SQL(96710784)
---------- --------------------------------------
1         CREATE OR REPLACE PROCEDURE xx  AS
BEGIN
SELECT SF_GET_SESSION_SQL(96710784);
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
```

33) SF_CLOB_LEN_IS_VALID

定义：

```
SF_CLOB_LEN_IS_VALID (
clob
)
```

功能说明：

检查系统存储的 clob 字符长度是否正常

参数说明：

clob：clob 对象

返回值：

0：不正常，1：正常

举例说明：

```
select SF_CLOB_LEN_IS_VALID ('PRODUCTION.PRODUCT.DESCRIPTION');
```

34) SP_VALIDATE_CLOB_LEN

定义：

SP_VALIDATE_CLOB_LEN(
clob
)

功能说明：

修复系统存储的 clob 字符长度

参数说明：

clob：clob 对象

举例说明：

```
SP_VALIDATE_CLOB_LEN ('PRODUCTION.PRODUCT.DESCRIPTION');
```

35) CHECK_INDEX_PAGE_USED

定义：

CHECK_INDEX_PAGE_USED (
  indexid   int
)

功能说明：

检查 ID 为 indexid 的索引数据页（包含 BLOB 字段）分配是否与对应的簇分配情况一致。

参数说明：

Indexid：索引 ID，如果不是数据库中的索引 ID 或者为空，则报错

返回值：

1：一致

0：不一致

举例说明：

```
DROP TABLE T1_CHECK;
CREATE TABLE T1_CHECK(c1 INT);

SELECT CHECK_INDEX_PAGE_USED(a.id) FROM sysobjects a WHERE a.subtype$='INDEX' AND a.pid IN(SELECT id FROM sysobjects WHERE name = 'T1_CHECK');
```

36) SF_FILE_SYS_CHECK_REPORT

定义：

INTEGER
SF_FILE_SYS_CHECK_REPORT(
  ts_id   int
)

功能说明：

校验检查指定表空间的簇是否正常

参数说明：

ts_id：指定检测的表空间，如果表空间不存在，返回 0

返回值：

1：表示表空间的簇都是正常的

0：表空间中存在检验不通过的簇，问题的详细描述输出到服务器的运行日志中。

举例说明：

```
SELECT SF_FILE_SYS_CHECK_REPORT(4);
```

37) SP_LOAD_LIC_INFO()

定义：

VOID
SP_LOAD_LIC_INFO()

功能说明：

进行 DM 服务器的 LICENSE 校验，检查 LICENSE 与当前 DM 版本及系统运行环境是否一致，如果有不一致则 DM 服务器主动退出。

参数说明：

无

返回值：

无

举例说明：

```
SP_LOAD_LIC_INFO();
```

38) SP_CREATE_DYNAMIC_TABLES()

定义：

```
VOID
SP_CREATE_DYNAMIC_TABLES (
  flag          int
)
```

功能说明：

创建或删除所有动态视图及其同义词。

参数说明：

flag：0 表示删除所有动态视图及其同义词；1 表示创建所有动态视图及其同义词，并授权。

返回值：

无

举例说明：

```
SP_CREATE_DYNAMIC_TABLES(1);
```

3) 备份恢复管理

1. SF_BAKSET_BACKUP_DIR_ADD

定义：

```
INT
SF_BAKSET_BACKUP_DIR_ADD(
    device_type varchar,
```

```
    backup_dir varchar(256)
)
```

功能说明：

添加备份目录。若添加目录已经存在或者为库默认备份路径，则认为已经存在，不添加，但也不报错。

参数说明：

device_type：待添加的备份目录对应存储介质类型，DISK 或者 TAPE。目前，无论指定介质类型为 DISK 或者 TAPE，都会同时搜索两种类型的备份集。

backup_dir：待添加的备份目录。

返回值：

1：目录添加成功；其它情况下报错。

举例说明：

```
SELECT SF_BAKSET_BACKUP_DIR_ADD('DISK','/home/dm_bak');
```

2. SF_BAKSET_BACKUP_DIR_REMOVE

定义：

```
INT
SF_BAKSET_BACKUP_DIR_REMOVE (
    device_type varchar,
    backup_dir varchar(256)
)
```

功能说明：

删除备份目录。若删除目录为库默认备份路径，不进行删除，认为删除失败。若指定目录存在于记录的合法目录中，则删除；不存在或者为空则跳过，正常返回。

参数说明：

device_type：待删除的备份目录对应存储介质类型。待删除的备份目录对应存储介质类型，DISK 或者 TAPE。

backup_dir：待删除的备份目录。

返回值：

1：目录删除成功、目录不存在或者目录为空；0：目录为库默认备份路径；其他情况报错。

举例说明：

```
SELECT SF_BAKSET_BACKUP_DIR_REMOVE('DISK','/home/dm_bak');
```

3. SF_BAKSET_BACKUP_DIR_REMOVE_ALL

定义：

```
INT
SF_BAKSET_BACKUP_DIR_REMOVE_ALL()
```

功能说明：

清理全部备份目录，默认备份目录除外。

返回值：

1：目录全部清理成功；其它情况下报错。

举例说明：

```
SELECT SF_BAKSET_BACKUP_DIR_REMOVE_ALL();
```

4. SF_BAKSET_CHECK

定义：

```
INT
SF_BAKSET_CHECK(
    device_type varchar,
    bakset_pathvarchar(256)
)
```

功能说明：

对备份集进行校验。

参数说明：

device_type：设备类型，disk 或 tape。

bakset_path：待校验的备份集目录。

返回值：

1：备份集目录存在且合法；否则报错。

举例说明：

```
BACKUP DATABASE FULL BACKUPSET '/home/dm_bak/db_bak_for_check';
SELECT SF_BAKSET_CHECK('DISK','/home/dm_bak/ db_bak_for_check');
```

5. SF_BAKSET_REMOVE

定义：

```
INT
SF_BAKSET_REMOVE (
    device_type varchar,
    backsetpath varchar(256),
    option integer
)
```

功能说明：

删除指定设备类型和指定备份集目录的备份集。一次只检查一个合法.meta 文件，然后删除对应备份集；若存在非法或非正常备份的.meta 文件，则报错或直接返回，不会接着检查下一个.meta 文件；若同一个备份集下还存在其它备份文件或备份集，则只删除备份文件，不会删除整个备份集。

参数说明：

device_type：设备类型，disk 或 tape。

backsetpath：待删除的备份集目录。

Option:删除备份集选项，0 默认删除，1 级联删除。可选参数。并行备份集中子备份集不允许单独删除。目标备份集被其他备份集引用为基备份的，默认删除，报错；级联删除情况下，会递归将相关的增量备份也删除。

返回值：

1：备份集目录删除成功，其它情况下报错。

举例说明：

```
BACKUP DATABASE FULL BACKUPSET '/home/dm_bak/db_bak_for_remove';
BACKUP DATABASE INCREMENT BACKUPSET
'/home/dm_bak/db_bak_for_remove_incr';
```

```
SELECT SF_BAKSET_REMOVE('DISK','/home/dm_bak/db_bak_for_remove');
```

执行结果

```
[-8202]:[/home/dm_bak/db_bak_for_remove_incr]的基备份，不能删除.
```

```
SELECT SF_BAKSET_REMOVE('DISK','/home/dm_bak/db_bak_for_remove',1);
```

执行结果

```
1
```

6. SF_BAKSET_REMOVE_BATCH

定义：

```
INT
SF_BAKSET_REMOVE_BATCH (
    device_type varchar,
    end_time    datetime,
    range       int,
    obj_name    varchar(257)
)
```

功能说明：

批量删除满足指定条件的所有备份集。

参数说明：

device_type：设备类型，disk 或 tape。指定 NULL，则忽略存储设备的区分。

end_time：删除备份集生成的结束时间，仅删除 end_time 之前的备份集，必须指定。

range：指定删除备份的级别。1 代表库级，2 代表表空间级，3 代表表级，4 代表归档备份。若指定 NULL，则忽略备份集备份级别的区分。

obj_name：待删除备份集中备份对象的名称，仅表空间级和表级有效。若为表级备份删除，则需指定完整的表名（模式.表名），否则，将认为删除会话当前模式下的表备份。若指定为 NULL，则忽略备份集中备份对象名称区分。

返回值：

1：备份集目录删除成功，其它情况下报错。

举例说明：

```
BACKUP DATABASE FULL BACKUPSET '/home/dm_bak/db_bak_for_remove';
BACKUP TABLESPACEMAIN FULL BACKUPSET '/home/dm_bak/ts_bak_for_remove';
SELECT SF_BAKSET_REMOVE_BATCH ('DISK', now(), NULL, NULL);
```

7. SP_DB_BAKSET_REMOVE_BATCH

定义：

```
SP_DB_BAKSET_REMOVE_BATCH (
    device_type varchar,
    end_time    datetime
)
```

功能说明：

批量删除指定时间之前的数据库备份集。使用该方法前，需要先使用 SF_BAKSET_BACKUP_DIR_ADD 添加将要删除的备份集目录，否则只删除默认备份路径下的备份集。

参数说明：

device_type：设备类型，disk 或 tape。指定 NULL，则忽略存储设备的区分。

end_time：删除备份集生成的结束时间，仅删除 end_time 之前的备份集，必须指定。

举例说明：

```
BACKUP DATABASE FULL BACKUPSET '/home/dm_bak/db_bak_for_batch_del';
SELECT SF_BAKSET_BACKUP_DIR_ADD('DISK','/home/dm_bak');
CALL SP_DB_BAKSET_REMOVE_BATCH('DISK', NOW());
```

8. SP_TS_BAKSET_REMOVE_BATCH

定义：

```
SP_TS_BAKSET_REMOVE_BATCH (
    device_type   varchar,
    end_time      datetime,
    ts_name       varchar(128)
)
```

功能说明：

批量删除指定表空间对象及指定时间之前的表空间备份集。使用该方法前，需要先使用 SF_BAKSET_BACKUP_DIR_ADD 添加将要删除的备份集目录，否则只删除默认备份路径下的备份集。

参数说明：

device_type：设备类型，disk 或 tape。指定 NULL，则忽略存储设备的区分。

end_time：删除备份集生成的结束时间，仅删除 end_time 之前的备份集，必须指定。

ts_name：表空间名，若未指定，则认为删除所有满足条件的表空间备份集。

举例说明：

```
BACKUP TABLESPACE MAIN BACKUPSET '/home/dm_bak/ts_bak_for_batch_del';
SELECT SF_BAKSET_BACKUP_DIR_ADD('DISK','/home/dm_bak');
CALL SP_TS_BAKSET_REMOVE_BATCH('DISK',NOW(),'MAIN');
```

9. SP_TAB_BAKSET_REMOVE_BATCH

定义：

```
SP_TAB_BAKSET_REMOVE_BATCH (
    device_type varchar,
    end_time    datetime,
    sch_name varchar(128),
    tab_name varchar(128)
)
```

功能说明：

批量删除指定表对象及指定时间之前的表备份集。使用该方法前，需要先使用 SF_BAKSET_BACKUP_DIR_ADD 添加将要删除的备份集目录，否则只删除默认备份路径下的备份集。

参数说明：

device_type：设备类型，disk 或 tape。指定 NULL，则忽略存储设备的区分。

end_time：删除备份集生成的结束时间，仅删除 end_time 之前的备份集，必须指定。

sch_name：表所属的模式名。

tab_name：表名，只要模式名和表名有一个指定，就认为需要匹配目标；若均指定为 NULL，则认为删除满足条件的所有表备份。

举例说明：

```
CREATE TABLE TAB_FOR_BATCH_DEL(C1 INT);
BACKUP TABLE TAB_FOR_BATCH_DEL BACKUPSET'/home/dm_bak/tab_bak_for_batch_del';
SELECT SF_BAKSET_BACKUP_DIR_ADD('DISK','/home/dm_bak');
CALL SP_TAB_BAKSET_REMOVE_BATCH('DISK',NOW(),'SYSDBA','TAB_FOR_BATCH_DEL');
```

10. SP_ARCH_BAKSET_REMOVE_BATCH

定义：

```
SP_ARCH_BAKSET_REMOVE_BATCH (
    device_type varchar,
    end_time    datetime
)
```

功能说明：

批量删除指定时间之前的归档备份集。使用该方法前，需要先使用 SF_BAKSET_BACKUP_DIR_ADD 添加将要删除的备份集目录，否则只删除默认备份路径下的备份集。

参数说明：

device_type：设备类型，disk 或 tape。指定 NULL，则忽略存储设备的区分。

end_time：删除备份集生成的结束时间，仅删除 end_time 之前的备份集，必须指定。

举例说明：

```
BACKUP ARCHIVELOG BACKUPSET '/home/dm_bak/arch_bak_for_batch_del';
SELECT SF_BAKSET_BACKUP_DIR_ADD('DISK','/home/dm_bak');
CALL SP_ARCH_BAKSET_REMOVE_BATCH('DISK', NOW());
```

4）定时器管理

本小节中的定时器管理相关系统存储过程都必须在系统处于 MOUNT 状态时执行。

1. SP_ADD_TIMER*

定义：

```
SP_ADD_TIMER(
    TIMER_NAME            VARCHAR(128),
    TYPE                  INT,
    FREQ_MONTH_WEEK_INTERVAL    INT,
    FREQ_SUB_INTERVAL         INT,
    FREQ_MINUTE_INTERVAL      INT,
    START_TIME                TIME,
    END_TIME                  TIME,
    DURING_START_DATE         DATETIME,
    DURING_END_DATE           DATETIME,
    NO_END_DATE_FLAG      BOOL
    DESCRIBE                  VARCHAR(128),
    IS_VALID                      BOOL
    )
```

功能说明：

创建一个定时器。

参数说明：

timer_name：定时器名称，应使用普通标识符，包含特殊符号可能导致无法正常使用。

type：定时器调度类型，取值如下：

1：执行一次
2：按日执行
3：按周执行
4：按月执行的第几天
5：按月执行的第一周
6：按月执行的第二周
7：按月执行的第三周
8：按月执行的第四周
9：按月执行的最后一周

freq_month_week_interval：间隔的月/周/天（调度类型决定）数

freq_sub_interval：第几天/星期几/一个星期中的某些天

freq_minute_interval：间隔的分钟数

start_time：开始时间

end_time：结束时间

during_start_date：有效日期时间段的开始日期时间。只有当前时间大于该字段时，该定时器才有效

during_end_date：有效日期时间段的结束日期时间

no_end_date_flag：是否有结束日期（0：有结束日期；1：没有结束日期）

describe：描述

is_valid：定时器是否有效

返回值：

无

说明：

1. type = 1 时，freq_sub_interval，freq_month_week_interval，freq_minute_interval，end_time，during_end_date 无效。只有 start_time，during_start_date 有意义。
2. type = 2 时，freq_month_week_interval 有效，表示相隔几天，取值范围为 1-100，freq_sub_interval 无效，freq_minute_interval <=24* 60 有效。
   a) 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效。
   b) 当 0<freq_minute_interval <=24*60 时,表示当天从 start_time 时间开始，每隔 freq_minute_interval 分钟执行一次。
   c) 当 freq_minute_interval > 24* 60 时，非法。
3. type = 3 时，意思是每隔多少周开始工作（从开始日期算起）。计算方法为：(当前日期和开始日期的天数之差/7)% freq_month_week_interval =0 且当前是星期 freq_sub_interval，其中 freq_sub_interval 的八位如下图所示：

8	7	6	5	4	3	2	1

   1-7 位分别代表星期天，星期一、星期二…、星期六，第 8 位无意义。这几位为 1 表示满足条件，为 0 表示不满足条件。
   a) 1<=freq_month_week_interval<=100，代表每隔多少周；
   b) 1<=freq_sub_interval<=127 代表星期中的某些天
   c) freq_minute_interval 代表分钟数。
      i. 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效
      ii. 当 freq_minute_interval > 0 且，表示当天可执行多次。
      iii. 当 freq_minute_interval > 24* 60 时，非法。
4. type = 4 时，每 freq_month_week_interval 个月的第 freq_sub_interval 日开始工作。其中，是否满足 freq_month_week_interval 个月的判断条件是：(月份的差 + (日期的差>=15||日期的差<= -15)?1:0) % freq_month_week_interval = 0 且当前是当月的 freq_sub_interval 日，表示满足条件；否则不满足。其中，“月份的差” 和 “日期的差”分别指的是系统当前时间和字段“during_start_date”中的月份差值和日期差值。
   a) 1<=freq_sub_interval<31，代表第几日；
   b) 1<=freq_month_week_interval<100，代表每隔多少个月；
   c) freq_minute_interval 代表分钟数。
      i. 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效
      ii. 当 freq_minute_interval > 0 且，表示当天可执行多次。
      iii. 当 freq_minute_interval >24* 60 时，非法。
5. type = 5，6，7，8，9 时，每 freq_month_week_interval 个月的第 type-4 周的周 freq_sub_interval 开始工作。其中，是否满足

freq_month_week_interval 个月的判断条件是：(月份的差 + 日期的差 /15) % freq_month_week_interval = 0 且当前是当月的第 type-4 周的周 freq_sub_interval，表示满足条件；否则不满足。

a) 1<=freq_sub_interval<=7，代表星期天到星期六（星期天是一个星期的第一天）；

b) freq_month_week_interval<100，代表每隔多少个月；

c) freq_minute_interval 代表分钟数。

  i. 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效。

  ii. 当 freq_minute_interval > 0 且，表示当天可执行多次。

  iii. 当 freq_minute_interval > 24* 60 时，非法。

6. 如果 no_end_date_flag = TRUE：表示永远不结束，一直存在下去。
7. 如果 is_valid= TRUE：表示定时器创建时就有效。
8. 总结 freq_sub_interval ， freq_month_week_interval 和 freq_minute_interval 的取值范围如下表所示。

type	1		2		3		4		5,6,7,8,9	
	max	min	max	min	max	min	max	min	max	min
freq_sub_interval	0	0	0	0	127	1	31	1	7	1
freq_month_week_interval	0	0	100	1	100	1	100	1	100	1
freq_minute_interval	0	0	1440	0	1440	0	1440	0	1440	0

举例说明：

创建一个定时器，每天 02：00 进行调度，开始日期：2011-02-01，结束日期：2011-09-01，间隔天数 1 天，每隔一分钟循环执行。

```
CALL SP_ADD_TIMER('TIMER1', 2, 1, 0, 1, '02:00:00', '20:00:00', '2011-02-01 14:30:34', '2011-09-01', 0, '每天凌晨两点进行调度', 1);
```

2. SP_DROP_TIMER*

定义：

```
SP_DROP_TIMER (
    timer_name varchar(128)
)
```

功能说明：

删除一个定时器。

参数说明：

timer_name：定时器名

返回值：

无

举例说明：

删除定时器 TIMER1

```
CALL SP_DROP_TIMER('TIMER1');
```

3. SP_OPEN_TIMER*

定义：

```
SP_OPEN_TIMER (
    timer_name varchar(128)
)
```

功能说明：

打开一个定时器。

参数说明：

timer_name：定时器名

返回值：

无

举例说明：

打开定时器 TIMER1

```
SP_OPEN_TIMER('TIMER1');
```

4. SP_CLOSE_TIMER*

定义：

```
SP_CLOSE_TIMER (
    timer_name varchar(128)
)
```

功能说明：

关闭一个定时器。

参数说明：

timer_name：定时器名

返回值：

无

举例说明：

关闭定时器 TIMER1

```
SP_CLOSE_TIMER('TIMER1');
```

5）数据复制管理

本小节的存储过程都与 DM 的数据复制功能相关，关于数据复制的概念和相关环境配置与操作可以参考《DM8 系统管理员手册》相关章节。

1. SP_INIT_REP_SYS*

定义：

```
SP_INIT_REP_SYS(
CREATE_FLAG      INT
);
```

功能说明：

创建或删除数据复制所需的系统表

参数说明：

CREATE_FLAG：为 1 表示创建复制所需系统表；为 0 表示删除这些系统表

返回值：

无

举例说明：

创建复制所需的系统表

```
SP_INIT_REP_SYS(1);
```

2. SP_RPS_ADD_GROUP

定义：

```
SP_RPS_ADD_GROUP(
    GROUP_NAME    VARCHAR(128),
    GROUP_DESC        VARCHAR(1000)
);
```

功能说明：

创建复制组

参数说明：

GROUP_NAME：创建的复制组名称

GROUP_DESC：复制组描述

返回值：

无

备注：

指示 RPS 创建一个新的复制组。如果已存在同名复制组则报错。

举例说明：

创建复制组 REP_GRP_B2C

```
SP_RPS_ADD_GROUP('REP_GRP_B2C','主从同步复制');
```

3. SP_RPS_DROP_GROUP

定义：

```
SP_RPS_DROP_GROUP(
    GROUP_NAME    VARCHAR(128)
);
```

功能说明：

删除复制组

参数说明：

GROUP_NAME：复制组名称

返回值：

无

举例说明：

删除复制组 REP_GRP_B2C

```
SP_RPS_DROP_GROUP ('REP_GRP_B2C');
```

4. SP_RPS_ADD_REPLICATION

定义：

```
SP_RPS_ADD_REPLICATION(
    GRP_NAME        VARCHAR(128),
    REP_NAME        VARCHAR(128),
    REP_DESC        VARCHAR(1000),
    MINSTANCE       VARCHAR(128),
    SINSTANCE       VARCHAR(128),
    REP_TIMER       VARCHAR(128),
    ARCH_PATH       VARCHAR(256)
);
```

功能说明：

创建复制关系

参数说明：

GRP_NAME：复制组名

REP_NAME：复制名，必须在 RPS 上唯一

REP_DESC：复制描述

MINSTANCE：主节点实例名，必须在 RPS 的 MAL 中已配置

SINSTANCE：从节点实例名，必须在 RPS 的 MAL 中已配置

REP_TIMER：复制定时器名。借助定时器，可以设置复制数据的同步时机。如果是同步复制则为 NULL

ARCH_PATH：主服务器上逻辑日志的完整归档路径。

返回值：

无

举例说明：

创建复制关系

```
SP_RPS_ADD_REPLICATION ('REP_GRP_B2C', 'REPB2C', 'B 到 C 的同步复制',
'B', 'C', NULL, '{ DEFARCHPATH}\REPB2C');
```

5. SP_RPS_DROP_REPLICATION

定义：

```
SP_RPS_DROP_REPLICATION (
    REP_NAME        VARCHAR(128)
);
```

功能说明：

删除复制关系

参数说明：

REP_NAME：复制名称

返回值：

无

举例说明：

删除复制关系

```
SP_RPS_DROP_REPLICATION ('REPB2C');
```

6. SP_RPS_SET_ROUTE_FAULT_TIMEOUT

定义：

```
SP_RPS_SET_ROUTE_FAULT_TIMEOUT (
    REP_NAME      VARCHAR(128),
    TIMEOUTS         INT
);
```

功能说明：

设置复制路径故障超时

参数说明：

REP_NAME：复制关系名。

TIMEOUTS：故障超时值，以秒为单位。0 为立即超时；-1 表示无超时限制

返回值：

无

备注：

该接口用于设置复制路径故障处理策略。设置后，RPS 如检测到复制路径产生故障，且故障持续超过设定的超时值后，则需要取消故障的复制关系。

举例说明：

设置复制路径故障超时

```
SP_RPS_SET_ROUTE_FAULT_TIMEOUT ('REPB2C',10);
```

7. SP_RPS_SET_INST_FAULT_TIMEOUT

定义：

```
SP_RPS_SET_INST_FAULT_TIMEOUT (
    INST_NAME     VARCHAR(128),
    TIMEOUTS         INT
);
```

功能说明：

设置复制节点故障超时

参数说明：

INST_NAME：复制节点实例名

TIMEOUTS：故障超时值，以秒为单位。0 为立即超时；-1 表示无超时限制

返回值：

无

举例说明：

设置复制节点故障超时

```
SP_RPS_SET_INST_FAULT_TIMEOUT ('B',10);
```

8. SP_RPS_ADD_TIMER

定义：

```
SP_RPS_ADD_TIMER(
    TIMER_NAME       VARCHAR(128),
    TIMER_DESC       VARCHAR(1000),
    TYPE$               INT,
```

```
        FERQ_INTERVAL        INT,
        FREQ_SUB_INTERVAL    INT,
        FREQ_MINUTE_INTERVAL    INT,
        START_TIME       TIME,
        END_TIME         TIME,
        DURING_START_DATE    DATETIME,
        DURING_END_DATE      DATETIME,
        NO_END_DATA_FLAG     INT
    );
```

功能说明：

设置复制关系的定时器

参数说明：

TIMER_NAME：定时器名

TIMER_DESC：定时器描述

TYPE$：定时器类型，取值如下：

1：执行一次

2：每日执行

3：每周执行

4：按月执行的第几天

5：按月执行的第一周

6：按月执行的第二周

7：按月执行的第三周

8：按月执行的第四周

9：按月执行的最后一周

FREQ_INTERVAL：间隔的月/周（调度类型决定）数

FREQ_SUB_INTERVAL： 间隔天数

FREQ_MINUTE_INTERVAL：间隔的分钟数

START_TIME：开始时间

END_TIME：结束时间

DURING_START_DATE：有效日期时间段的开始日期时间

DURING_END_DATE：有效日期时间段结束日期时间

NO_END_DATA_FLAG：结束日期是否无效标识，0 表示结束日期有效，1 表示无效

本过程的 TYPE$、FERQ_INTERVAL、FREQ_SUB_INTERVAL、FREQ_MINUTE_INTERVAL、START_TIME、END_TIME、DURING_START_DATE、DURING_END_DATE 和 NO_END_DATA_FLAG 分别与过程 SP_ADD_TIMER 的参数 TYPE、FREQ_MONTH_WEEK_INTERVAL、FREQ_SUB_INTERVAL、FREQ_MINUTE_INTERVAL、START_TIME、END_TIME、DURING_START_DATE、DURING_END_DATE 和 NO_END_DATE_FLAG 对应，其具体说明可参考过程 SP_ADD_TIMER 的说明。

返回值：

无

举例说明：

设置复制关系的定时器

```
        SP_RPS_ADD_TIMER ('TIMER1','按天计算', 1, 1, 0, 1, CURTIME, '23:59:59', 
NOW, NULL, 1);
```

9. SP_RPS_REP_RESET_TIMER

定义：

```
SP_RPS_REP_RESET_TIMER(
    REP_NAME      VARCHAR(128),
    TIMER_NAME        VARCHAR(128)
);
```

功能说明：

重新设置复制关系的定时器

参数说明：

REP_NAME：复制名

TIMER_NAME：新的定时器名

返回值：

无

举例说明：

重新设置复制关系的定时器

```
SP_RPS_REP_RESET_TIMER ('REPB2C', 'TIMER1');
```

10. SP_RPS_ADD_TAB_MAP

定义：

```
SP_RPS_ADD_TAB_MAP(
    REP_NAME      VARCHAR(128),
    MTAB_SCHEMA   VARCHAR(128),
    MTAB_NAME VARCHAR(128),
    STAB_SCHEMA   VARCHAR(128),
    STAB_NAME     VARCHAR(128),
    READ_ONLY_MODE      INT
);
```

功能说明：

添加表级复制映射

参数说明：

REP_NAME：复制关系名

MTAB_SCHEMA：主表模式名

MTAB_NAME：主表名

STAB_SCHEMA：从表模式名

STAB_NAME：从表名

READ_ONLY_MODE：只读复制模式，1 表示只读模式，从表只接受复制更新，0 表示非只读模式

返回值：

无

举例说明：

添加复制映射

```
SP_RPS_ADD_TAB_MAP('REPB2C', 'USER1', 'T1', 'USER2', 'T2', 0);
```

11. SP_RPS_DROP_TAB_MAP

定义：

```
SP_RPS_DROP_TAB_MAP(
    REP_NAME      VARCHAR(128),
    MTAB_SCHEMA   VARCHAR(128),
    MTAB_NAME VARCHAR(128),
    STAB_SCHEMA   VARCHAR(128),
    STAB_NAME     VARCHAR(128),
);
```

功能说明：

删除表级复制映射

参数说明：

REP_NAME：复制关系名

MTAB_SCHEMA：主表模式名

MTAB_NAME：主表名

STAB_SCHEMA：从表模式名

STAB_NAME：从表名

返回值：

无

举例说明：

删除表级复制映射

```
SP_RPS_DROP_TAB_MAP('REPB2C', 'USER1', 'T1', 'USER2', 'T2');
```

12. SP_RPS_ADD_SCH_MAP

定义：

```
SP_RPS_ADD_SCH_MAP(
    REP_NAME      VARCHAR(128),
    MSCH          VARCHAR(128),
    SSCH          VARCHAR(128),
    READ_ONLY_MODE      INT
);
```

功能说明：

添加模式级复制映射

参数说明：

REP_NAME：复制关系名

MSCH：主模式名

SSCH： 从表模式名

READ_ONLY_MODE：只读复制模式，1 表示只读模式，从表只接受复制更新，0 表示非只读模式

返回值：

无

举例说明：

添加复制映射

```
SP_RPS_ADD_SCH_MAP('REPB2C', 'USER1', 'USER2', 0);
```

13. SP_RPS_DROP_SCH_MAP

定义：

```
SP_RPS_DROP_SCH_MAP(
    REP_NAME      VARCHAR(128),
    MSCH          VARCHAR(128),
    SSCH          VARCHAR(128)
);
```

功能说明：

删除模式级复制映射

参数说明：

REP_NAME：复制关系名

MSCH：主模式名

SSCH：从模式名

返回值：

无

举例说明：

删除模式级复制映射

```
SP_RPS_DROP_SCH_MAP('REPB2C', 'USER1', 'USER2');
```

14. SP_RPS_ADD_DB_MAP

定义：

```
SP_RPS_ADD_DB_MAP(
    REP_NAME      VARCHAR(128),
    READ_ONLY_MODE      INT
);
```

功能说明：

添加库级复制映射

参数说明：

REP_NAME：复制关系名

READ_ONLY_MODE：只读复制模式，1 表示只读模式，从表只接受复制更新，0 表示非只读模式

返回值：

无

举例说明：

添加库级复制映射

```
SP_RPS_ADD_DB_MAP('REPB2C', 0);
```

15. SP_RPS_DROP_DB_MAP

定义：

```
SP_RPS_DROP_DB_MAP(
```

```
        REP_NAME        VARCHAR(128)
    );
```

功能说明：

删除库级复制映射

参数说明：

REP_NAME：复制关系名

返回值：

无

举例说明：

删除库级复制映射

```
SP_RPS_DROP_DB_MAP('REPB2C');
```

16. SP_RPS_SET_BEGIN

定义：

```
SP_RPS_SET_BEGIN(
    GRP_NAME        VARCHAR(128),
);
```

功能说明：

开始复制设置

参数说明：

GRP_NAME：复制组名

返回值：

无

备注：

开始对指定复制组进行属性设置。创建/删除复制关系与创建/删除复制映射等接口都必须在此接口调用后执行，否则会报错"错误的复制设置序列"。同一会话中也不能同时开始多个复制设置。

举例说明：

复制组 REPB2C 开始复制

```
SP_RPS_SET_BEGIN(' REP_GRP_B2C');
```

17. SP_RPS_SET_APPLY

定义：

```
SP_RPS_SET_APPLY ();
```

功能说明：

提交复制设置，保存并提交本次设置的所有操作。如果需要继续设置，则必须重新调用 SP_RPS_SET_BEGIN

参数说明：

无

返回值：

无

举例说明：

提交复制设置

```
SP_RPS_SET_APPLY ();
```

18. SP_RPS_SET_CANCEL

定义：

SP_RPS_SET_CANCEL ();

功能说明：

放弃复制设置，放弃本次设置的所有操作。如果需要重新设置，则必须再次调用 SP_RPS_SET_BEGIN

参数说明：

无

返回值：

无

举例说明：

放弃复制设置

```
SP_RPS_SET_CANCEL();
```

6）模式对象相关信息管理

1. SP_TABLEDEF

定义：

```
void
SP_TABLEDEF (
    schname   varchar(128),
    tablename  varchar(128)
)
```

功能说明：

以结果集的形式返回表的定义

参数说明：

schname：模式名

tablename：表名

返回值：

无

举例说明：

```
CALL  SP_TABLEDEF('PRODUCTION','PRODUCT');
```

2. SP_VIEWDEF

定义：

```
void
SP_VIEWDEF (
    schname   varchar(128),
    viewname  varchar(128)
)
```

功能说明：

以结果集的形式返回视图的定义

参数说明：

schname：模式名

viewname：视图名

返回值：

无

举例说明：

```
CALL  SP_VIEWDEF('PURCHASING','VENDOR_EXCELLENT');
```

3. SF_VIEW_EXPIRED

定义：

```
INI*
SF_VIEW_EXPIRED(
SCHNAME varchar(128),
VIEWNAME varchar(128)
)
```

功能说明：

检查当前系统表中视图列定义是否有效

返回值：

返回 0 或 1。0 表示有效，1 表示无效

举例说明：

```
CREATE TABLE T_01_VIEW_DEFINE_00(C1 INT,C2 INT);
CREATE VIEW TEST_T_01_VIEW_DEFINE_00 AS SELECT* FROM
T_01_VIEW_DEFINE_00;
SELECT SF_VIEW_EXPIRED('SYSDBA','TEST_T_01_VIEW_DEFINE_00');
--查询结果为 0

ALTER TABLE T_01_VIEW_DEFINE_00 DROP COLUMN C1 CASCADE ;
SELECT SF_VIEW_EXPIRED('SYSDBA','TEST_T_01_VIEW_DEFINE_00');
--查询结果为 1
```

4. CHECKDEF

定义：

```
CHAR *
CHECKDEF (
    consid    int,
    preflag   int
)
```

功能说明：

获得 check 约束的定义

参数说明：

consid：check 约束 id 号

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

check 约束的定义

举例说明：

```
CREATE TABLE TEST_CHECKDEF(C1 INT CHECK(C1>10));
```

```
    通过查询系统表
    SELECT A.name, A.ID FROM SYSOBJECTS A, SYSOBJECTS B WHERE B.NAME='TEST_CHECKDEF' AND A.PID=B.ID AND A.SUBTYPE$='CONS';
    --得到约束 ID 为 134217770
    SELECT CHECKDEF(134217770,1);
```

5. CONSDEF

定义：

```
CHAR *
CONSDEF (
    indexid int,
    preflag  int
)
```

功能说明：

获取 unique 约束的定义

参数说明：

indexid：索引号数字字符串

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

unique 约束的定义

举例说明：

```
    CREATE TABLE TEST_CONSDEF(C1 INT PRIMARY KEY,C2 INT, CONSTRAINT CONS1 UNIQUE(C2));
    --通过查询系统表
    SELECT C.INDEXID FROM SYSOBJECTS O,SYSCONS C WHERE O.NAME='CONS1' AND O.ID=C.ID;
    --系统生成 C2 上的 INDEX 为 33555481
    SELECT CONSDEF(33555481,1);
```

6. INDEXDEF

定义：

```
CHAR *
INDEXDEF (
    indexid  int,
    preflag  int
)
```

功能说明：

获取 index 的创建定义

参数说明：

indexid：索引 ID

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

索引的创建定义

举例说明：

```
        CREATE INDEX PRODUCT_IND ON PRODUCTION.PRODUCT(PRODUCTID);
        --查询系统表得到索引 ID
        SELECT NAME, ID FROM SYSOBJECTS WHERE NAME='PRODUCT_IND' AND
SUBTYPE$='INDEX';
        SELECT indexdef(33555530,1);
```

7. SP_REORGANIZE_INDEX

定义：

```
SP_REORGANIZE_INDEX (
    schname    varchar(128),
    indexname  varchar(128)
)
```

功能说明：

对指定索引进行空间整理

参数说明：

schname：模式名

indexname：索引名

返回值：

无

举例说明：

```
CREATE INDEX PRODUCT_IND ON PRODUCTION.PRODUCT(PRODUCTID);
CALL SP_REORGANIZE_INDEX('PRODUCTION','PRODUCT_IND');
```

8. SP_REBUILD_INDEX

定义：

```
SP_REBUILD_INDEX (
    schname    varchar(128),
    indexed      int
)
```

功能说明：

重建索引。约束：1. 水平分区子表、临时表和系统表上建的索引不支持重建；2. 虚索引和聚集索引不支持重建

参数说明：

schname：模式名

indexid：索引 ID

返回值：

无

举例说明：

```
CREATE INDEX PRODUCT_IND ON PRODUCTION.PRODUCT(PRODUCTID);
--查询系统表得到索引 ID
SP_REBUILD_INDEX('SYSDBA', 33555530);
```

9. CONTEXT_INDEX_DEF

定义：

```
CHAR *
CONTEXT_INDEX_DEF (
    indexed   int,
    preflag   int
)
```

功能说明：

获取 context_index 的创建定义

参数说明：

indexid：索引 ID

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

索引的创建定义

举例说明：

```
CREATE CONTEXT INDEX PRODUCT_CIND ON PRODUCTION.PRODUCT(NAME) LEXER DEFAULT_LEXER;
--查询系统表得到全文索引 ID
SELECT NAME, ID FROM SYSOBJECTS WHERE NAME='PRODUCT_CIND';
SELECT CONTEXT_INDEX_DEF(33555531, 1);
```

10. SYNONYMDEF

定义：

```
CHAR *
SYNONYMDEF (
username varchar(128),
    synname  varchar(128),
    type     int,
    preflag   int
)
```

功能说明：

获取同义词的创建定义

参数说明：

username：用户名

synname：同义词名

type：同义词类型。 0，public 1，user

preflag：对象前缀个数。1 表示导出模式名，0 表示只导出对象名

返回值：

同义词的创建定义

举例说明：

```
SELECT SYNONYMDEF('SYSDBA', 'SYSOBJECTS',0,1);
```

11. SEQDEF

定义：

```
CHAR *
SEQDEF (
```

```
    seqid    int,
    preflag    int
)
```

功能说明：

获取序列的创建定义

参数说明：

seqid：序列 id 号

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

序列的创建定义

举例说明：

```
CREATE SEQUENCE SEQ1;
SELECT ID FROM SYSOBJECTS WHERE NAME='SEQ1'; --查出 id 为 167772160
SELECT SEQDEF(167772160, 1);
```

12. IDENT_CURRENT

定义：

```
int
IDENT_CURRENT (
    fulltablename varchar(8187)
)
```

功能说明：

获取自增列当前值

参数说明：

fulltablename：表全名；格式为"模式名.表名"

返回值：

自增列当前值

举例说明：

```
SELECT IDENT_CURRENT('PRODUCTION.PRODUCT');
```

13. IDENT_SEED

定义：

```
INT
IDENT_SEED (
        fulltablename varchar(8187)
)
```

功能说明：

获取自增列种子

参数说明：

fulltablename：表全名；格式为"模式名.表名"

返回值：

自增列种子

举例说明：

```
SELECT IDENT_SEED('PRODUCTION.PRODUCT');
```

14. IDENT_INCR

定义：

```
INT
IDENT_INCR (
    fulltablename varchar(8187)
    )
```

功能说明：

获取自增列增量值 increment

参数说明：

fulltablename：表全名；格式为"模式名.表名"

返回值：

自增列增量值 increment

举例说明：

```
SELECT IDENT_INCR('PRODUCTION.PRODUCT');
```

15. SCOPE_IDENTITY

定义：

```
INT SCOPE_IDENTITY ();
```

功能说明：

返回插入到同一作用域中的 identity 列内的最后一个 identity 值

返回值：

RVAL：函数返回值，长度为 8

举例说明：

详见GLOBAL_IDENTITY例子

16. GLOBAL_IDENTITY

定义：

```
INT GLOBAL_IDENTITY();
```

功能说明：

返回在当前会话中的任何表内所生成的最后一个标识值，不受限于特定的作用域。一个作用域就是一个模块：存储过程、触发器、函数或批处理，若两个语句处于同一个存储过程、函数或批处理中，则它们位于相同的作用域中

返回值：

RVAL：函数返回值，长度为 8

举例说明：

```
DROP TABLE T1;
DROP TABLE T2;
CREATE TABLE T1(C1 INT IDENTITY(1,1), C2 CHAR);
CREATE TABLE T2(C1 INT IDENTITY(1,1), C2 CHAR);
INSERT INTO T1(C2) VALUES('a');
INSERT INTO T1(C2) VALUES('b');
INSERT INTO T1(C2) VALUES('c');
COMMIT;
SELECT SCOPE_IDENTITY();
--返回值：3
SELECT GLOBAL_IDENTITY();
--返回值：3
CREATE OR REPLACE TRIGGER TRI1 AFTER INSERT ON T1
```

```
BEGIN
    INSERT INTO T2(C2) VALUES('a');
END;

INSERT INTO T1(C2) VALUES('d');
COMMIT;

SELECT SCOPE_IDENTITY();
--返回值：4
SELECT GLOBAL_IDENTITY();
--返回值：1
```

17. SF_CUR_SQL_STR

定义：

```
clob
SF_CUR_SQL_STR (
        IS_TOP INT
)
```

功能说明：

用于并行环境中，获取当前执行的 SQL 语句

参数说明：

IS_TOP：取 0 时返回当前层计划执行的语句，取 1 时返回顶层计划语句

返回值：

sql 语句

举例说明：

```
SELECT SF_CUR_SQL_STR(0);
```

18. SF_COL_IS_CHECK_KEY

定义：

```
INT
SF_COL_IS_CHECK_KEY (
    key_num    int,
    key_info   varchar(8187),
    col_id     int
    )
```

功能说明：

判断一个列是否为CHECK约束列

参数说明：

key_num：约束列总数；

key_info：约束列信息；

col_id：列id

返回值：

返回1：表示该列是check约束列，否则返回0。

举例说明：

```
CREATE TABLE TC (C1 INT, C2 DOUBLE, C3 DATE, C4 VARCHAR, CHECK(C1 <
100 AND C4 IS NOT NULL));
SELECT TBLS.NAME, COLS.NAME, COLS.COLID, COLS.TYPE$, COLS.LENGTH$,
COLS.SCALE, COLS.NULLABLE$, COLS.DEFVAL
```

```
            FROM    (SELECT ID, NAME FROM SYS.SYSOBJECTS WHERE NAME = 'TC' AND
TYPE$ = 'SCHOBJ' AND SUBTYPE$ = 'UTAB'  AND SCHID = (SELECT ID FROM SYS.SYSOBJECTS
WHERE NAME = 'SYSDBA' AND TYPE$ = 'SCH')) AS TBLS,
            (SELECT ID, PID, INFO1, INFO6 FROM SYS.SYSOBJECTS WHERE TYPE$ =
'TABOBJ' AND SUBTYPE$ = 'CONS') AS CONS_OBJ,SYS.SYSCOLUMNS AS COLS,SYS.SYSCONS
AS CONS WHERE  TBLS.ID = CONS_OBJ.PID AND TBLS.ID    =    COLS.ID    AND
    SF_COL_IS_CHECK_KEY(CONS_OBJ.INFO1, CONS_OBJ.INFO6, COLS.COLID) = 1 AND
    CONS.TABLEID = TBLS.ID AND CONS.TYPE$ = 'C';
```

19. SF_REPAIR_HFS_TABLE

定义：

```
SF_REPAIR_HFS_TABLE (
    SCHNAME  VARCHAR(128),
    TABNAME  VARCHAR(128)
    )
```

功能说明：

HUGE表日志属性为LOG NONE时，如果系统出现故障，导致该表数据不一致，则通过该函数修复表数据，保证数据的一致性。

参数说明：

SCHNAME：模式名；

TABNAME：表名；

返回值：

成功返回0，否则报错。

举例说明：

```
CREATE HUGE TABLE T_DM(C1 INT, C2 VARCHAR(20)) LOG NONE;
INSERT INTO T_DM VALUES(99, 'DM8');
COMMIT;
UPDATE T_DM SET C1 = 100;--系统故障
SF_REPAIR_HFS_TABLE('SYSDBA','T_DM');
```

20. SP_ENABLE_EVT_TRIGGER

定义：

```
SP_ENABLE_EVT_TRIGGER (
    SCHNAME   varchar(128),
    TRINAME   varchar(128),
    ENABLE BOOL
    )
```

功能说明：

禁用/启用指定的事件触发器。

参数说明：

SCHNAME：模式名；

TABNAME：触发器名；

ENABLE： 1 表示启用，0 表示禁用。

返回值：

成功返回0，否则报错。

举例说明：

```
SP_ENABLE_EVT_TRIGGER('SYSDBA', 'TRI_1', 1);
SP_ENABLE_EVT_TRIGGER('SYSDBA', 'TRI_1', 0);
```

21. SP_ENABLE_ALL_EVT_TRIGGER

定义：

```
SP_ENABLE_ALL_EVT_TRIGGER (
    enable bool
    )
```

功能说明：

禁用/启用数据库上的所有事件触发器。

参数说明：

ENABLE：1 表示启用，0 表示禁用。

返回值：

成功返回0，否则报错。

举例说明：

```
SP_ENABLE_ALL_EVT_TRIGGER(1);
SP_ENABLE_ALL_EVT_TRIGGER(0);
```

22. SF_GET_TRIG_DEPENDS

定义：

```
SF_GET_TRIG_DEPENDS(
TRIGID int
    )
```

功能说明：

查看触发器的依赖项。

参数说明：

TRIGID：触发器id。

返回值：

触发器的依赖项。

举例说明：

```
CREATE TABLE T1_TRI_10000(OBJECTTYPE VARCHAR);
CREATE TRIGGER TRI1_TRI_10000 BEFORE CREATE ON DATABASE BEGIN INSERT INTO T1_TRI_10000 VALUES(:EVENTINFO.OBJECTTYPE); END;
SELECT SF_GET_TRIG_DEPENDS((SELECT ID FROM SYSOBJECTS WHERE NAME LIKE 'TRI1_TRI_10000'));
```

23. SF_GET_PROC_DEPENDS

定义：

```
SF_GET_PROC_DEPENDS(
PROCID
 int
    )
```

功能说明：

查看过程/函数的依赖项。

参数说明：

PROCID：过程/函数id。

返回值：

过程/函数的依赖项。

举例说明：

```
CREATE OR REPLACE FUNCTION FUN1 (A INT) RETURN INT AS
       S  INT;
BEGIN
  S:=A*A;
  RETURN S;
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
/

CREATE OR REPLACE FUNCTION FUN2 (A INT, B INT) RETURN INT AS
       S  INT;
BEGIN
  S:=A+B+FUN1(A);
  RETURN S;
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
/


SELECT SF_GET_PROC_DEPENDS((SELECT ID FROM SYSOBJECTS WHERE NAME LIKE
'FUN2'));
```

7) 数据守护管理

本小节的存储过程都与 DM 的数据守护功能相关，关于数据守护的概念和相关环境配置与操作可以参考《DM8 数据守护与读写分离集群 V4.0》相关章节。

1. SP_SET_OGUID

定义：

```
void
SP_SET_OGUID (
    oguid   int
    )
```

功能说明：

设置主备库监控组的 ID 号

参数说明：

oguid：oguid

返回值：

无

举例说明：

```
SP_SET_OGUID (451245);
```

2. SF_GET_BROADCAST_ADDRESS

定义：

```
VARCHAR
SF_GET_BROADCAST_ADDRESS (
IP_ADDR       VARCHAR,
SUBNET_MASK   VARCHAR
)
```

功能说明：

根据 IPv4 的 IP 地址以及子网掩码计算广播地址

参数说明：

IP_ADDR：输入的 IP 地址

SUBNET_MASK：子网掩码地址

返回值：

广播地址

示例：

```
SELECT SF_GET_BROADCAST_ADDRESS('15.16.193.6','255.255.248.0');
```

3. SP_SET_RT_ARCH_VALID

定义：

```
VOID SP_SET_RT_ARCH_VALID ()
```

功能说明：

设置实时归档有效。

参数说明：

无

返回值：

无

举例说明：

```
CALL SP_SET_RT_ARCH_VALID ();
```

4. SP_SET_RT_ARCH_INVALID

定义：

```
VOID  SP_SET_RT_ARCH_INVALID ()
```

功能说明：

设置实时归档无效。

参数说明：

无

返回值：

无

举例说明：

```
CALL SP_SET_RT_ARCH_INVALID ();
```

5. SF_GET_RT_ARCH_STATUS

定义：

```
VARCHAR  SF_GET_RT_ARCH_STATUS ()
```

功能说明：

获取实时归档状态。

参数说明：

无

返回值：

有效：VALID。无效：INVALID

举例说明：

```
SELECT SF_GET_RT_ARCH_STATUS ();
```

6. SP_SET_ARCH_STATUS

定义：

```
VOID
SP_SET_ARCH_STATUS (
 ARCH_DEST  VARCHAR,
 ARCH_STATUS INT
)
```

功能说明：

设置指定归档目标的归档状态

参数说明：

ARCH_DEST：归档目标名称。

ARCH_STATUS：要设置的归档状态，取值 0 或 1，0 表示有效状态，1 表示无效状态。

返回值：

无

举例说明：

```
CALL SP_SET_ARCH_STATUS('DM1', 1);
```

7. SP_SET_ALL_ARCH_STATUS

定义：

```
VOID
SP_SET_ALL_ARCH_STATUS (
ARCH_STATUS INT
)
```

功能说明：

设置所有归档目标的归档状态

参数说明：

ARCH_STATUS：要设置的归档状态，取值 0 或 1，0 表示有效状态，1 表示无效状态。

返回值：

无

举例说明：

```
CALL SP_SET_ALL_ARCH_STATUS(1);
```

8. SP_APPLY_KEEP_PKG

定义：

```
VOID  SP_APPLY_KEEP_PKG()
```

功能说明：

APPLY 备库的 KEEP_PKG 数据。

参数说明：

无

返回值：

无

举例说明：

```
CALL SP_APPLY_KEEP_PKG();
```

9. SP_DISCARD_KEEP_PKG

定义：

VOID SP_DISCARD_KEEP_PKG()

功能说明：

丢弃备库的 KEEP_PKG 数据。

参数说明：

无

返回值：

无

举例说明：

```
CALL SP_DISCARD_KEEP_PKG();
```

10. SP_CLEAR_ARCH_SEND_INFO

定义：

VOID SP_CLEAR_ARCH_SEND_INFO()

VOID SP_CLEAR_ARCH_SEND_INFO(INST_NAMEVARCHAR)

功能说明：

此系统函数在主库上执行有效，用于清理到备库最近 N 次的归档发送信息。

如果不指定 INST_NAME，则清理所有备库最近 N 次的归档发送信息。

N 值为 V$ARCH_SEND_INFO 中的 RECNT_SEND_CNT 值。

参数说明：

INST_NAME:备库实例名

返回值：

无

举例说明：

```
CALL SP_CLEAR_ARCH_SEND_INFO();
CALL SP_CLEAR_ARCH_SEND_INFO('GRP1_RWW_02');
```

11. SP_CLEAR_RAPPLY_STAT

定义：

VOID SP_CLEAR_RAPPLY_STAT()

功能说明：

此系统函数在备库上执行有效，用于清理此备库最近 N 次的日志重演信息。

N 值为 V$RAPPLY_STAT 中的 RECNT_APPLY_NUM 值。

参数说明：

无

返回值：

无

举例说明：

```
CALL SP_CLEAR_RAPPLY_STAT();
```

12. SP_ADD_TIMELY_ARCH(

定义：

```
VOID  SP_ADD_TIMELY_ARCH(
INST_NAME VARCHAR
)
```

功能说明：

OPEN 状态下动态扩展 TIMELY 归档。

参数说明：

INST_NAME：TIMELY 归档的目标实例名

返回值：

无

举例说明：

```
CALL SP_ADD_TIMELY_ARCH('DB3');
```

13. SF_MAL_CONFIG

定义：

```
    VOID
SF_MAL_CONFIG(
CFG_FLAG      INT,
    BRO_FLAG      INT
)
```

功能说明：

设置 MAL 配置状态。

参数说明：

CFG_FLAG：1，设置配置状态；0 取消配置状态

BRO_FLAG：1，多节点广播设置；0 本地设置

返回值：

无

举例说明：

```
SF_MAL_CONFIG(1,0);
```

14. SF_MAL_INST_ADD

定义：

```
VOID
SF_MAL_INST_ADD(
    ITEM_NAME     VARCHAR,
    INST_NAME     VARCHAR,
    MAL_IP        VARCHAR,
```

MAL_PORT INT,
MAL_INST_IP VARCHAR,
MAL_INST_PORT INT
)

功能说明：

增加 MAL 配置项

参数说明：

ITEM_NAME: 配置项名称
INST_NAME: 实例名
MAL_IP: MAL IP 地址
MAL_PORT: MAL 端口
MAL_INST_IP 实例 IP 地址
MAL_INST_PORT 实例端口

举例说明：

```
SF_MAL_INST_ADD('MAL_INST3','EP03','192.168.0.14', 5378, '192.168.0.14',5238);
```

15. SF_MAL_CONFIG_APPLY

定义：

VOID
SF_MAL_CONFIG_APPLY()

功能说明：

将 MAL 配置生效。

参数说明：

无。

举例说明：

```
SF_MAL_CONFIG_APPLY();
```

8) MPP 管理

1. SP_SET_SESSION_MPP_SELECT_LOCAL

定义：

VOID
SP_SET_SESSION_MPP_SELECT_LOCAL (
local_flag int
)

功能说明：

MPP 系统下设置当前会话是否只查询本节点数据。如果不设置，表示可以查询全部节点数据。

参数说明：

local_flag: 设置标记。1 代表只查询本节点数据；0 代表查询全部节点数据。

返回值：

无

举例说明：

```
SP_SET_SESSION_MPP_SELECT_LOCAL(1);
```

2. SF_GET_SESSION_MPP_SELECT_LOCAL

定义：

INT SF_GET_SESSION_MPP_SELECT_LOCAL ()

功能说明：

查询 MPP 系统下当前会话是否只查询本节点数据

参数说明：

无

返回值：

1 代表只查询本节点数据，0 查询全部节点数据

举例说明：

```
SELECT SF_GET_SESSION_MPP_SELECT_LOCAL();
```

3. SP_SET_SESSION_LOCAL_TYPE

定义：

```
void
    SP_SET_SESSION_LOCAL_TYPE (
    DDL_FLAG      int
)
```

功能说明：

MPP 下本地登录时，设置本会话上是否允许 DDL 操作。本地登录默认不允许 DDL 操作。

参数说明：

DDL_FLAG：为 1 时表示允许当前本地会话执行 DDL 操作，为 0 时则不允许。

返回值：

无

举例说明：

MPP 下本地登录会话

```
SP_SET_SESSION_LOCAL_TYPE (1);
CREATE TABLE TEST(C1 INT);
SP_SET_SESSION_LOCAL_TYPE (0);
```

4. SF_GET_EP_SEQNO

定义：

```
INT
SF_GET_EP_SEQNO (
rowid     bigint
)
```

功能说明：

根据查询出的行数据的 ROWID 获取本条数据来自哪个 EP 站点

参数说明：

rowid：行数据的 ROWID

返回值：

MPP 系统内 EP 站点序号

举例说明：

```
SELECT SF_GET_EP_SEQNO(ROWID) FROM V$INSTANCE;
```

5. SF_GET_SELF_EP_SEQNO

定义：

```
INT
SF_GET_SELF_EP_SEQNO ()
```

功能说明：

获取本会话连接的 EP 站点序号

参数说明：

无

返回值：

获取本会话连接的 EP 站点序号

举例说明：

```
SELECT SF_GET_SELF_EP_SEQNO();
```

6. SP_GET_EP_COUNT

定义：

```
INT
SP_GET_EP_COUNT (
SCH_NAME VARCHAR(128),
TAB_NAME VARCHAR(128)
);
```

功能说明：

统计 MPP 环境下表在各个站点的数据行数

参数说明：

SCH_NAME：表所在模式名

TAB_NAME：表名

举例说明：

```
SP_GET_EP_COUNT('SYSDBA','T');
```

7. SF_MPP_INST_ADD

定义：

```
VOID
SF_MPP_INST_ADD(
    ITEM_NAME     VARCHAR,
        INST_NAME     VARCHAR,
)
```

功能说明：

增加 MPP 实例配置。

参数说明：

ITEM_NAME：配置项名称

INST_NAME：实例名

举例说明：

```
SF_MPP_INST_ADD(' SERVICE_NAME3', 'EP03');
```

8. SF_MPP_INST_REMOVE

定义：

```
VOID
SF_MPP_INST_REMOVE(
        INST_NAME     VARCHAR,
)
```

功能说明：

删除 MPP 实例。

参数说明：

INST_NAME：实例名

举例说明：

```
SF_MPP_INST_REMOVE('EP03');
```

9) 日志与检查点管理

1. CHECKPOINT

定义：

```
INT
CHECKPOINT (
 rate   int
)
```

功能说明：

设置检查点。需要注意的是：在 DSC 环境下，OK 节点个数大于 1 时，MOUNT 状态下调用该函数不会生效。

参数说明：

rate：刷脏页百分比，取值范围：1~100

返回值：

检查点是否成功，0 表示成功，非 0 表示失败

举例说明：

设置刷脏页百分比为 30%的检查点

```
SELECT CHECKPOINT(30);
```

2. SF_ARCHIVELOG_DELETE_BEFORE_TIME

定义：

```
INT
SF_ARCHIVELOG_DELETE_BEFORE_TIME (
 time   datetime
)
```

功能说明：

数据库以归档模式打开的情况下，删除指定时间之前的归档日志文件。待删除

的文件必须处于未被使用状态。

参数说明：

time：指定删除的最大关闭时间，若大于当前使用归档日志文件的创建时间，则从当前使用归档文件之前的归档日志文件开始删除。

返回值：

删除归档日志文件数，-1 表示出错。

举例说明：

删除三天之前的归档日志。

```
SELECT SF_ARCHIVELOG_DELETE_BEFORE_TIME(SYSDATE - 3);
```

3. SF_ARCHIVELOG_DELETE_BEFORE_LSN

定义：

```
INT
SF_ARCHIVELOG_DELETE_BEFORE_LSN (
 lsn   bigint
)
```

功能说明：

数据库以归档模式打开的情况下，删除小于指定 LSN 值的归档日志文件。待删除的文件必须处于未被使用状态。

参数说明：

lsn：指定删除的最大 LSN 值文件，若指定 lsn 值大于当前正在使用归档日志的起始 LSN（arch_lsn），则从当前使用归档文件之前的文件开始删除。

返回值：

删除归档日志文件数，-1 表示出错。

举例说明：

删除 LSN 值小于 95560 的归档日志文件。

```
SELECT SF_ARCHIVELOG_DELETE_BEFORE_LSN(95560);
```

10) 数据库重演

1. SP_START_CAPTURE

定义：

```
SP_START_CAPTURE(
    path      varchar(256),
    duration  int
)
```

功能说明：

手工设置负载捕获开始

参数说明：

path：捕获文件保存的绝对路径

duration：捕获持续的时间，如果设置为-1 秒，表示需要手动停止捕获，或者磁盘空间满了自动停止。

返回值：

无

2. SP_STOP_CAPTURE

定义：

SP_STOP_CAPTURE()

功能说明：

手工停止数据库重演捕获

参数说明：

无

返回值：

无

举例说明：

首先对数据库进行备份，然后调用 SP_START_CAPTURE，创建一张表，插入 3 条数据，最后调用 SP_STOP_CAPTURE。利用重演工具 dreplay 进行数据库重演。

```
CALL SP_START_CAPTURE('D:\dmdbms\data', -1);

CREATE TABLE "SYSDBA"."TEST4REPLAY"
(
    C1 INT,
    C2 VARCHAR(64)
);

INSERT INTO "SYSDBA"."TEST4REPLAY" VALUES(1, 'A');
INSERT INTO "SYSDBA"."TEST4REPLAY" VALUES(2, 'B');
INSERT INTO "SYSDBA"."TEST4REPLAY" VALUES(3, 'C');
COMMIT;

CALL SP_STOP_CAPTURE();
--然后利用 dreplay 工具进行数据库重演：
dreplay SERVER=localhost:5236 FILE=D:\dmdbms\data
```

11) 统计信息

以下对象不支持统计信息：1. 外部表、DBLINK 远程表、动态视图、记录类型数组所用的临时表；2. 所在表空间为 OFFLINE 的对象；3. 位图索引，位图连接索引、虚索引、无效的索引、全文索引；4. BLOB、IMAGE、LONGVARBINARY、CLOB、TEXT、LONGVARCHAR 等列类型。

1. SP_TAB_INDEX_STAT_INIT*

定义：

```
SP_TAB_INDEX_STAT_INIT (
    schname    varchar(128),
    tablename   varchar(128)
```

)

功能说明：

对表上所有的索引生成统计信息

参数说明：

schname：模式名

tablename：表名

举例说明：

对 SYSOBJECTS 表上所有的索引生成统计信息

```
CALL SP_TAB_INDEX_STAT_INIT ('SYS', 'SYSOBJECTS');
```

2. SP_DB_STAT_INIT*

定义：

```
SP_DB_STAT_INIT (
)
```

功能说明：

对库上所有模式下的所有用户表以及表上的所有索引生成统计信息

举例说明：

对库上所有模式下的所有用户表以及表上的所有索引生成统计信息

```
CALL SP_DB_STAT_INIT ();
```

3. SP_INDEX_STAT_INIT*

定义：

```
SP_INDEX_STAT_INIT (
    schname    varchar(128),
    indexname  varchar(128)
)
```

功能说明：

对指定的索引生成统计信息

参数说明：

schname：模式名

indexname：索引名

举例说明：

对指定的索引 IND 生成统计信息

```
CALL SP_INDEX_STAT_INIT ('SYSDBA', 'IND');
```

4. SP_COL_STAT_INIT*

定义：

```
SP_COL_STAT_INIT (
    schname    varchar(128),
    tablename    varchar(128),
    colname    varchar(128)
```

)

功能说明：

对指定的列生成统计信息，不支持大字段列和虚拟列

参数说明：

schname：模式名

tablename：表名

colname：列名

举例说明：

对表 SYSOBJECTS 的 ID 列生成统计信息

```
CALL SP_COL_STAT_INIT ('SYS', 'SYSOBJECTS', 'ID');
```

5. SP_TAB_COL_STAT_INIT*

定义：

```
SP_TAB_COL_STAT_INIT (
    schname    varchar(128),
    tablename   varchar(128)
)
```

功能说明：

对某个表上所有的列生成统计信息

参数说明：

schname：模式名

tablename：表名

举例说明：

对'SYSOBJECTS'表上所有的列生成统计信息

```
CALL SP_TAB_COL_STAT_INIT ('SYS', 'SYSOBJECTS');
```

6. SP_STAT_ON_TABLE_COLS

定义：

```
SP_STAT_ON_TABLE_COLS(
    SCHEMA_NAME VARCHAR(128),
    TABLE_NAME VARCHAR(128),
    E_PERCENT INT
)
```

功能说明：

对某个表上所有的列，按照指定的采样率生成统计信息

参数说明：

schname：模式名

tablename：表名

E_PERCENT：采样率（0-100]

举例说明：

对'SYSOBJECTS'表上所有的列生成统计信息，采样率 90

```
CALL SP_STAT_ON_TABLE_COLS ('SYS','SYSOBJECTS',90);
```

7. SP_TAB_STAT_INIT*

定义：

```
SP_TAB_STAT_INIT (
    schname    varchar(128),
    tablename   varchar(128)
)
```

功能说明：

对某张表或某个索引生成统计信息

参数说明：

schname：模式名

tablename：表名或索引名

举例说明：

对表 SYSOBECTS 生成统计信息

```
CALL SP_TAB_STAT_INIT ('SYS', 'SYSOBJECTS');
```

8. SP_SQL_STAT_INIT*

定义：

```
SP_SQL_STAT_INIT (
    sql    varchar(8187)
)
```

功能说明：

对某个 SQL 查询语句中涉及的所有表和过滤条件中的列(不包括大字段、ROWID)生成统计信息。

可能返回的错误提示：

1） 语法分析出错，sql 语句语法错误

2） 对象不支持统计信息，统计的表或者列不存在，或者不允许被统计

参数说明：

sql：sql 语句

举例说明：

对'SELECT * FROM SYSOBJECTS'语句涉及的所有表生成统计信息

```
CALL SP_SQL_STAT_INIT ('SELECT * FROM SYSOBJECTS');
```

9. SP_INDEX_STAT_DEINIT*

定义：

```
SP_INDEX_STAT_DEINIT (
    schname    varchar(128),
    indexname  varchar(128)
)
```

功能说明：

清空指定索引的统计信息

参数说明：

schname：模式名

indexname：索引名

举例说明：

清空索引 IND 的统计信息

```
CALL SP_INDEX_STAT_DEINIT ('SYSDBA', 'IND');
```

10. SP_COL_STAT_DEINIT*

定义：

```
SP_COL_STAT_DEINIT (
    schname    varchar(128),
    tabname    varchar(128),
    colname    varchar(128)
)
```

功能说明：

删除指定列的统计信息

参数说明：

schname：模式名

tabname：表名

colname：列名

举例说明：

删除 SYSOBJECTS 的 ID 列的统计信息

```
CALL SP_COL_STAT_DEINIT ('SYS', 'SYSOBJECTS', 'ID');
```

11. SP_TAB_COL_STAT_DEINIT*

定义：

```
SP_TAB_COL_STAT_DEINIT (
    schname    varchar(128),
    tablename   varchar(128)
)
```

功能说明：

删除表上所有列的统计信息

参数说明：

schname：模式名

tablename：表名

举例说明：

删除 SYSOBJECTS 表上所有列的统计信息

```
CALL SP_TAB_COL_STAT_DEINIT ('SYS', 'SYSOBJECTS');
```

12. SP_TAB_STAT_DEINIT*

定义：

```
SP_TAB_STAT_DEINIT (
```

```
    schname     varchar(128),
    tablename   varchar(128)
)
```

功能说明：

删除某张表的统计信息

参数说明：

schname：模式名

tablename：表名

举例说明：

删除表 SYSOBECTS 的统计信息

```
CALL SP_TAB_STAT_DEINIT ('SYS', 'SYSOBJECTS');
```

13. ET

定义：

```
ET(
ID_IN INT
);
```

功能说明：

统计执行 ID 为 ID_IN 的所有操作符的执行时间。需设置 INI 参数 ENABLE_MONITOR=1、MONITOR_TIME=1 和 MONITOR_SQL_EXEC=1。

参数说明：

ID_IN：SQL 语句的执行 ID

举例说明：

```
select count(*) from sysobjects where name='SYSDBA';
行号        COUNT(*)
---------- --------------------
1          2

已用时间: 14.641(毫秒). 执行号:26.
```

可以得到执行号为 26。

```
et(26);
```

得到结果：

```
OP    TIME(US)        PERCENT RANK                SEQ         N_ENTER
----- ------------ -------------------- ----------- -----------
DLCK   4                0.14%   6                  0           2
PRJT2  15               0.53%   5                  2           4
AAGR2  78               2.74%   4                  3           5
NSET2  128              4.49%   3                  1           3
SLCT2  465              16.32%  2                  4           7
SSCN   2160             75.79%  1                  5           4
```

结果中每一行对应一个操作符，每一列对应的解释如下：

列名	含义
OP	操作符名称
TIME(US)	执行耗时，以微秒为单位

PERCENT RANK	在整个计划中用时占比
SEQ	计划中的序号
N_ENTER	操作符进入的次数

如果某些操作符并没有真正执行或者耗时很短，则不会在 ET 中显示。另外，结果中出现了 EXPLAIN 计划中没有的 DLCK 操作符，它是用来对字典对象的上锁处理，通常情况下可以忽略。

14. SP_UPDATE_SYSSTATS

定义：

```
SP_UPDATE_SYSSTATS (
flag    int
)
```

功能说明：

用以升级及迁移统计信息。7.1.5.173 及之后的版本支持该功能。

参数说明：

flag：标记，取值 0、1、2、3、4、99。 0：回退版本；1：创建统计信息升级辅助表； 2：升级 SYS.SYSSTATS 表结构； 3：导入 SYS.SYSSTATS 表数据； 4：删除升级辅助表；参数 99：理想状态下使用，集合了参数 1、2、3、4 的功能。

举例说明：

如果想要将老库升级，需要手动升级，升级步骤如下：

```
SP_UPDATE_SYSSTATS(99);
或者
SP_UPDATE_SYSSTATS(1);
SP_UPDATE_SYSSTATS(2);
SP_UPDATE_SYSSTATS(3);
SP_UPDATE_SYSSTATS(4);
```

如果想使用老库，且使用了之前的加列版本服务器且升级失败，那么需要先回退版本，再升级：

```
SP_UPDATE_SYSSTATS(0);
SP_UPDATE_SYSSTATS(99);
```

15. SP_TAB_MSTAT_DEINIT

定义：

```
SP_TAB_MSTAT_DEINIT(
schname    varchar(128),
tablename   varchar(128)
)
```

功能说明：

删除一个表的多维统计信息。

参数说明：

schname：模式名

tablename：表名

举例说明：

删除表 SYSDBA.L1 上所有的多维统计信息

```
CALL SP_TAB_MSTAT_DEINIT('SYSDBA','L1');
```

16. SF_GET_MD_COL_STR

定义：

```
SF_GET_MD_COL_STR(
tabid      int,
col_info   varbinary(4096),
col_num    int
)
```

功能说明：

获取表上建多维统计信息的列信息。

参数说明：

tabid：表 id

col_info：多维的维度信息

col_num：维度

返回值是：多维统计信息的列信息

举例说明：

获取表 L1 上建的多维统计信息的列信息

```
SELECT SF_GET_MD_COL_STR(ID, MCOLID, N_DIMENSION) FROM SYS.SYSMSTATS
WHERE ID IN (SELECT ID FROM SYSOBJECTS WHERE NAME='L1');
```

12) 资源监测

1. SP_CHECK_IDLE_MEM

定义：

```
SP_CHECK_IDLE_MEM (
)
```

功能说明：

对可用内存空间进行检测，并在低于阈值（对应 INI 参数 IDLE_MEM_THRESHOLD)的情况下打印报警记录到日志,同时报内存不足的异常。

举例说明：

监测当前系统的内存空间是否低于阀值

```
CALL  SP_CHECK_IDLE_MEM ();
```

2. SP_CHECK_IDLE_DISK

定义：

```
SP_CHECK_IDLE_DISK (
    path  varchar(256)
)
```

功能说明：

对指定位置的磁盘空间进行检测，并在低于阈值（对应 INI 参数 IDLE_DISK_THRESHOLD）的情况下打印报警记录到日志，同时报磁盘空间不足的异常。

参数说明：

path：监测的路径

举例说明：

监测 d:\data 路径下的磁盘空间是否低于阀值

```
CALL  SP_CHECK_IDLE_DISK ('d:\data');
```

3. SF_GET_CMD_RESPONSE_TIME

定义：

SF_GET_CMD_RESPONSE_TIME()

功能说明：

查看 DM 服务器对用户命令的平均响应时间

返回值：

命令的平均响应时间，单位秒

举例说明：

在 dm.ini 中 ENABLE_MONITOR 取值不小于 2 的前提下执行

```
SELECT SYS.SF_GET_CMD_RESPONSE_TIME();
```

4. SF_GET_TRX_RESPONSE_TIME

定义：

SF_GET_TRX_RESPONSE_TIME()

功能说明：

查看事务的平均响应时间

返回值：

事务的平均响应时间，单位秒

举例说明：

在 dm.ini 中 ENABLE_MONITOR 取值不为 0 的前提下执行

```
SELECT SYS.SF_GET_TRX_RESPONSE_TIME();
```

5. SF_GET_DATABASE_TIME_PER_SEC

定义：

SF_GET_DATABASE_TIME_PER_SEC()

功能说明：

查看数据库中用户态时间占总处理时间的比值

返回值：

用户态时间占总处理时间的比值，该比值越大表明用于 IO、事务等待等耗费的时间越少

举例说明：

在 dm.ini 中 ENABLE_MONITOR 取值不小于 2 的前提下执行

```
SELECT SYS.SF_GET_DATABASE_TIME_PER_SEC();
```

6. TABLE_USED_SPACE

定义：

```
INT *
TABLE_USED_SPACE (
    schname  varchar(256);
```

```
        tabname  varchar(256)
    )
```

功能说明：

获取指定表所占用的页数

参数说明：

schname：模式名，必须大写

tabname：表名，必须大写

返回值：

表所占用的页数

举例说明：

查看 SYSOJBECTS 的所占用的页数

```
SELECT  TABLE_USED_SPACE ('SYS','SYSOBJECTS');
```

查询结果：

```
32
```

7. HUGE_TABLE_USED_SPACE

定义：

```
INT *
HUGE_TABLE_USED_SPACE (
    schname  varchar(256);
    tabname  varchar(256)
)
```

功能说明：

获取指定 HUGE 表所占用的大小

参数说明：

schname：模式名，必须大写

tabname：huge 表名，必须大写

返回值：

HUGE 表所占用的大小，单位：M

举例说明：

查看 huge 表 test 所占用的大小

```
CREATE  HUGE TABLE TEST(A INT);
SELECT  HUGE_TABLE_USED_SPACE ('SYSDBA','TEST');
```

查询结果：

```
64
```

8. USER_USED_SPACE

定义：

```
INT *
USER_USED_SPACE (
    username  varchar(256)
)
```

功能说明：

获取指定用户所占用的页数，不包括用户占用的 HUGE 表页数

参数说明：

username：用户名

返回值：

用户所占用的页数

举例说明：

查看 SYSDBA 的所占用的页数

```
SELECT  USER_USED_SPACE ('SYSDBA');
```

查询结果：

```
64
```

9. TS_USED_SPACE

定义：

```
INT *
TS_USED_SPACE (
    tsname  varchar(256)
)
```

功能说明：

获取指定表空间所有文件所占用的页数之和。RLOG 和 ROLL 表空间不支持。

参数说明：

tsname：表空间名

返回值：

表空间占用的页数

举例说明：

查看 MAIN 表空间所有文件占用的页数之和。

```
SELECT  TS_USED_SPACE ('MAIN');
```

查询结果：

```
16384
```

10. DB_USED_SPACE

定义：

```
INT *
DB_USED_SPACE ()
```

功能说明：

获取整个数据库占用的页数

返回值：

整个数据库占用的页数

举例说明：

查看数据库所占用的页数

```
SELECT  DB_USED_SPACE ();
```

查询结果：

```
19712
```

11. INDEX_USED_SPACE

定义：

```
INT *
INDEX_USED_SPACE (
    indexid  int
)
```

功能说明：

根据索引 ID，获取指定索引所占用的页数

参数说明：

indexid：索引 ID

返回值：

索引占用的页数

举例说明：

查看索引号为 33554540 的索引所占用的页数

```
SELECT  INDEX_USED_SPACE (33554540);
```

查询结果：

```
32
```

12. INDEX_USED_SPACE

定义：

```
INT *
INDEX_USED_SPACE (
    schname   varchar(256),
    indexname varchar(256)
)
```

功能说明：

根据索引名获取指定索引占用的页数

参数说明：

schname：模式名

indexname：索引名

返回值：

索引占用的页数

举例说明：

查看 SYSDBA 模式下索引名为 INDEX_TEST 的索引占用的页数

```
SELECT  INDEX_USED_SPACE ('SYSDBA', 'INDEX_TEST');
```

查询结果：

```
14
```

13. INDEX_USED_SPACE

定义：

```
INT *
INDEX_USED_SPACE (
    schname   varchar(256),
    tablename varchar(256),
    nth_index int
)
```

功能说明：

获取表指定序号的索引占用的页数

参数说明：

schname：模式名

tablename：表名

nth_index：要获取表的第几个索引

返回值：

索引占用的页数

举例说明：

查看 SYSDBA 模式下，TEST 表的第二个索引占用的页数

```
SELECT  INDEX_USED_SAPCE ('SYSDBA', 'TEST', 2);
```

查询结果：

```
14
```

14. INDEX_USED_PAGES

定义：

```
INT *
INDEX_USED_PAGES (
    indexid  int
)
```

功能说明：

根据索引 id，获取指定索引已使用的页数

参数说明：

indexid：索引 ID

返回值：

索引已使用的页数

举例说明：

查看索引号为 33554540 的索引已使用的页数

```
SELECT  INDEX_USED_PAGES (33554540);
```

查询结果：

```
14
```

15. INDEX_USED_PAGES

定义：

```
INT *
INDEX_USED_PAGES (
    schname   varchar(256),
    indexname varchar(256)
)
```

功能说明：

根据索引名，获取指定索引已使用的页数

参数说明：

schname：模式名

indexname：索引名

返回值：

索引占用的页数

举例说明：

查看 SYSDBA 模式下索引名为 INDEX_TEST 的索引已使用的页数

```
SELECT  INDEX_USED_PAGES ('SYSDBA', 'INDEX_TEST');
```

查询结果：

```
14
```

16. INDEX_USED_PAGES

定义：

```
INT *
INDEX_USED_PAGES (
    schname   varchar(256),
    tablename varchar(256),
    nth_index int
)
```

功能说明：

根据表的索引序号，获取指定索引已使用的页数

参数说明：

schname：模式名

tablename：表名

nth_index：要获取表的第几个索引

返回值：

索引占用的页数

举例说明：

查看 SYSDBA 模式下，TEST 表的第二个索引已使用的页数

```
SELECT  INDEX_USED_PAGES ('SYSDBA', 'TEST', 2);
```

查询结果：

```
14
```

17. TABLE_USED_PAGES

定义：

```
INT *
TABLE_USED_PAGES (
    schname  varchar(256);
    tabname  varchar(256)
)
```

功能说明：

获取指定表已使用的页数

参数说明：

schname：模式名

tabname：表名

返回值：

表已使用的页数

举例说明：

查看 SYSOJBECTS 已使用的页数

```
SELECT  TABLE_USED_PAGES('SYS','SYSOBJECTS');
```

查询结果：

```
14
```

18. TS_FREE_SPACE

定义：

```
INT *
TS_FREE_SPACE (
tsname  varchar(256)
)
```

功能说明：

获取指定表空间可分配的空闲页数。RLOG 和 ROLL 表空间不支持。

参数说明：

tsname：表空间名

返回值：

表空间可分配的空闲页数，包含 TS_RESERVED_SPACE()的预留页数。

举例说明：

查看 MAIN 表空间可分配的空闲页数

```
SELECT  TS_FREE_SPACE ('MAIN');
```

查询结果：

```
8192
```

19. TS_RESERVED_SPACE

定义：

```
INT *
TS_RESERVED_SPACE (
tsname  varchar(256)
)
```

功能说明：

获取指定表空间系统预留的页数。RLOG 和 ROLL 表空间不支持。系统启动时根据 dm.ini 参数 TS_RESERVED_EXTENTS 预留部分空间，避免 ROLLBACK/PURGE 等操作过程中分配数据页失败，这些预留空间用户无法使用。

参数说明：

tsname：表空间名

返回值：

表空间系统预留的页数

举例说明：

查看 MAIN 表空间系统预留的页数

```
SELECT  TS_RESERVED_SPACE ('MAIN');
```

查询结果：

```
512
```

20. TS_FREE_SPACE_CALC

定义：

```
INT *
TS_FREE_SPACE_CALC (
tsname  varchar(256)
)
```

功能说明：

重新计算指定表空间可分配的空闲页数。RLOG 和 ROLL 表空间不支持。

参数说明：

tsname：表空间名

返回值：

表空间可分配的空闲页数，包含 TS_RESERVED_SPACE 的预留页数。

举例说明：

重新计算 MAIN 表空间可分配的空闲页数

```
SELECT  TS_FREE_SPACE_CALC('MAIN');
```

查询结果：

```
8192
```

21. TABLE_USED_LOB_PAGES

定义：

```
INT *
TABLE_USED_LOB_PAGES (
schname  varchar(256);
tabname  varchar(256)
)
```

功能说明：

获取指定表已使用的 lob 页数，包括大字段列使用的行外数据 lob 页和指定了 USING LONG ROW 存储选项时变长字符串列使用的行外数据 lob 页

参数说明：

schname：模式名

tabname：表名

返回值：

表已使用的 lob 页数

举例说明：

查看 SYS.SYSTEXTS 表已使用的 lob 页数

```
SELECT TABLE_USED_LOB_PAGES('SYS','SYSTEXTS');
```

查询结果：

```
139
```

22. SP_SET_SQL_STAT_THRESHOLD

定义：

```
SP_SET_SQL_STAT_THRESHOLD(
name VARCHAR,
val INT64)
```

功能说明：

设置语句级资源监控，SQL 监控项生成的条件阀值，当资源大于设置的阀值时

才生成统计项。可以设置的监控项为 VSQL_STAT/VSQL_STAT_HISTORY 视图中的 5~58 列。该过程只有 DBA 权限才能执行。

参数说明：

name：允许为 NULL。需要设置阈值的监控项名字，可选名字为 VSQL_STAT/VSQL_STAT_HISTORY 视图中第 5~58 列的列名。当 name 为 NULL 时，val 只能设置为 0 或者-1。0 表示无条件生成历史监控项，-1 表示不生成历史监控项。

val：不允许为 NULL 且必须大于等于 0。表示需要设置的阈值。

返回值：

无

举例说明：

```
SP_SET_SQL_STAT_THRESHOLD('INS_IN_PL_CNT', 60000);
```

23. INDEX_USED_PAGES

定义：

```
int
INDEX_USED_PAGES (
schema_name   varchar(128),
index_name    varchar(128)
)
```

功能说明：

根据模式下的索引名获取指定索引已使用的页数。

参数说明：

schema_name：模式名

index_name：索引名

返回值：

指定索引已使用的页数

举例说明：

查看模式 SYSDBA 下索引名为 IDX1 的索引已使用的页数

```
SELECT INDEX_USED_PAGES('SYSDBA','IDX1');
```

24. INDEX_USED_PAGES

定义：

```
int
INDEX_USED_PAGES (
schema_name   varchar(128),
table_name    varchar(128),
index_no      int
)
```

功能说明：

根据模式下的指定表的第 index_no 个索引获取该索引使用的页数。

参数说明：

schema_name：模式名

table_name：表名

index_no：索引序号(从 1 开始)

返回值：

指定索引已使用的页数

举例说明：

查看模式 SYSDBA 下表名为 TAB1 的第一个索引已使用的页数

```
SELECT INDEX_USED_PAGES('SYSDBA','TAB1',1);
```

25. TABLE_ROWCOUNT

定义：

```
int
TABLE_ROWCOUNT (
schema_name   varchar(128),
table_name    varchar(128)
)
```

功能说明：

获取指定模式下指定表的总行数

参数说明：

schema_name：模式名

table_name：表名

返回值：

指定表的总行数

举例说明：

查看模式 SYSDBA 下表 TAB1 的总行数

```
SELECT TABLE_ROWCOUNT('SYSDBA','TAB1');
```

13) 类型别名

DM 支持用户对各种基础数据类型定义类型别名，定义的别名可以在 SQL 语句和 DMSQL 程序中使用。

1. SP_INIT_DTYPE_SYS*

定义：

```
SP_INIT_DTYPE_SYS (
        create_flag   int
)
```

功能说明：

初始化或清除类型别名运行环境

参数说明：

create_flag：1 表示创建，0 表示删除

返回值：

无

举例说明：

初始化类型别名运行环境

```
CALL SP_INIT_DTYPE_SYS(1);
```

2. SF_CHECK_DTYPE_SYS

定义：

```
int
SF_CHECK_DTYPE_SYS ()
```

功能说明：

系统类型别名系统的启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统类型别名系统的启用状态

```
SELECT SF_CHECK_DTYPE_SYS;
```

3. SP_DTYPE_CREATE*

定义：

```
SP_DTYPE_CREATE (
    name      varchar(32),
    base_name varchar(32),
    len       int,
    scale         int
)
```

功能说明：

创建一个类型别名。最多只能创建 100 个类型别名

参数说明：

name：类型别名的名称

base_name：基础数据类型名

len：基础数据类型长度，当为固定精度类型时，应该设置为 NULL

scale：基础数据类型刻度

返回值：

无

举例说明：

创建 VARCHAR(100)的类型别名‘STR’

```
CALL SP_DTYPE_CREATE('STR', 'VARCHAR', 100, NULL);
```

4. SP_DTYPE_DELETE*

定义：

```
SP_DTYPE_DELETE (
    name      varchar(32)
)
```

功能说明：

删除一个类型别名

参数说明：

name：类型别名的名称

返回值：

无

举例说明：

删除类型别名‘STR’

```
CALL SP_DTYPE_DELETE('STR');
```

14) 杂类函数

1. TO_DATETIME

定义：

```
CHAR *
TO_DATETIME(
    year int,
    month int,
    day int,
    hour int,
    minute int
)
```

功能说明：

将 int 类型值组合并转换成日期时间类型

参数说明：

year：年份，必须介于-4713 和+9999 之间，且不为 0

month：月份

day：日

hour：小时

minute：分钟

返回值：

日期时间值

举例说明：

将整型数 2010，2，2，5，5 转换成日期时间类型

```
SELECT  TO_DATETIME (2010,2,2,5,5);
```

查询结果：

```
2010-02-02 05:05:00.000000
```

2. ROWIDTOCHAR

定义：

```
VARCHAR
ROWIDTOCHAR (
    ROWID int(8)
)
```

功能说明：

将 INT 类型的 ROWID 值转换成 CHAR 类型

参数说明：

ROWID：int 类型的 rowid 值

返回值：

char 类型的 rowid

举例说明：

将 int 类型 rowid 转化为 char 类型 rowid，与 like 谓词中的字符条件进行匹配。

```
SELECT ROWID FROM sysobjects  WHERE ROWIDTOCHAR(ROWID) LIKE '_55' ORDER BY ROWID;
```

查询结果：

```
155
255
355
455
555
655
755
855
955
```

3. SP_SET_ROLE*

定义：

```
int
SP_SET_ROLE (
    ROLE_NAME   varchar(128),
    ENABLE      int
)
```

功能说明：

设置角色启用禁用

参数说明：

ROLE_NAME：角色名。

ENABLE：0/1 （0：禁用 1：启用）

返回值：

无

举例说明：

禁用角色 ROLE1

```
SP_SET_ROLE ( 'ROLE1', 0);
```

4. SP_CREATE_SYSTEM_VIEWS*

定义

```
int
SP_CREATE_SYSTEM_VIEWS (
CREATE_FLAG      int
)
```

功能说明：

创建或删除系统视图。详细的系统视图请参考《DM8 系统管理员手册》附录 3。

参数说明：

CREATE_FLAG：为 1 时表示创建视图；为 0 表示删除视图

返回值：

无

举例说明：

创建系统视图

```
SP_CREATE_SYSTEM_VIEWS(1);
```

5. SF_CHECK_SYSTEM_VIEWS

定义：

```
int
SF_CHECK_SYSTEM_VIEWS()
```

功能说明：

系统视图的启用状态检测。

返回值：

0：未启用；1：已启用

举例说明：

获得系统视图的启用状态

```
SELECT SF_CHECK_SYSTEM_VIEWS;
```

6. SP_DYNAMIC_VIEW_DATA_CLEAR

定义：

```
void
SP_DYNAMIC_VIEW_DATA_CLEAR (
    VIEW_NAME   varchar(128)
)
```

功能说明：

清空动态性能视图的历史数据，仅对存放历史记录的动态视图起作用。

参数说明：

VIEW_NAME：动态性能视图名

返回值：

无

举例说明：

清空动态性能视图 V$SQL_HISTORY 的历史数据

```
SP_DYNAMIC_VIEW_DATA_CLEAR('V$SQL_HISTORY');
```

7. SP_INIT_DBG_SYS*

定义：

```
void
SP_INIT_DBG_SYS(
    CREATE_FLAG      int
)
```

功能说明：

创建或删除 DBMS_DBG 系统包

参数说明：

CREATE_FLAG：为 1 时表示创建 DBMS_DBG 包；为 0 表示删除该系统包

返回值：

无

举例说明：

创建 DBMS_DBG 系统包

```
SP_INIT_DBG_SYS(1);
```

8. SF_CHECK_DBG_SYS

定义：

```
int
SF_CHECK_DBG_SYS ()
```

功能说明：

系统的 DBG 系统包启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得 DBG 系统包的启用状态

```
SELECT SF_CHECK_DBG_SYS;
```

9. SP_INIT_GEO_SYS*

定义：

```
void
SP_INIT_GEO_SYS(
    CREATE_FLAG      int
)
```

功能说明：

创建或删除 DMGEO 系统包

参数说明：

CREATE_FLAG：为 1 时表示创建 DMGEO 包；为 0 表示删除该系统包

返回值：

无

举例说明：

创建 DMGEO 系统包

```
SP_INIT_GEO_SYS(1);
```

10. SF_CHECK_GEO_SYS

定义：

```
int
SF_CHECK_GEO_SYS ()
```

功能说明：

系统的 GEO 系统包启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得 GEO 系统包的启用状态

```
SELECT SF_CHECK_GEO_SYS;
```

11. SP_CREATE_SYSTEM_PACKAGES*

定义：

```
void
SP_CREATE_SYSTEM_PACKAGES (
    CREATE_FLAG      int
)
```

功能说明：

创建或删除除了 DMGEO、DBMS_JOB 和 DBMS_WORKLOAD_REPOSITORY 以外的所有系统包。若在创建过程中某个系统包由于特定原因未能创建成功，会跳过继续创建后续的系统包。

参数说明：

CREATE_FLAG：为 1 时表示创建除了 DMGEO、DBMS_JOB 和 DBMS_WORKLOAD_REPOSITORY 以外的所有系统包；为 0 表示删除这些系统包

返回值：

无

举例说明：

创建除了 DMGEO、DBMS_JOB 和 DBMS_WORKLOAD_REPOSITORY 以外的所有系统包

```
SP_CREATE_SYSTEM_PACKAGES (1);
```

12. SP_CREATE_SYSTEM_PACKAGES*

定义：

```
void
SP_CREATE_SYSTEM_PACKAGES (
    CREATE_FLAG      int,
    PKGNAME          varchar(128)
)
```

功能说明：

创建或删除除了 DMGEO、DBMS_JOB 和 DBMS_WORKLOAD_REPOSITORY 以外的指定名字的系统包

参数说明：

CREATE_FLAG：为 1 时表示创建指定的系统包；为 0 表示删除这个系统包

PKGNAME：指定的除 DMGEO、DBMS_JOB 和 DBMS_WORKLOAD_REPOSITORY 以外的系统包名

返回值：

无

举例说明：

创建 DBMS_LOB 系统包

```
SP_CREATE_SYSTEM_PACKAGES(1, 'DBMS_LOB');
```

13. SF_CHECK_SYSTEM_PACKAGES

定义：

int

SF_CHECK_SYSTEM_PACKAGES ()

功能说明：

系统包的启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统包的启用状态

```
SELECT SF_CHECK_SYSTEM_PACKAGES;
```

14. SP_INIT_INFOSCH*

定义：

```
void
SP_INIT_INFOSCH (
    CREATE_FLAG      int
)
```

功能说明：

创建或删除信息模式

参数说明：

CREATE_FLAG：为 1 时表示创建信息模式；为 0 表示删除信息模式

举例说明：

创建信息模式

```
SP_INIT_INFOSCH (1);
```

15. SF_CHECK_INFOSCH

定义：

int

SF_CHECK_INFOSCH ()

功能说明：

系统的信息模式启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统信息模式的启用状态

```
SELECT SF_CHECK_INFOSCH;
```

16. SP_INIT_CPT_SYS

定义：

```
int
SP_INIT_CPT_SYS (
FLAG        int
)
```

功能说明：

初始化数据捕获环境

参数说明：

FLAG： 1 表示初始化环境；0 表示删除环境

举例说明：

初始化数据捕获环境

```
SP_INIT_CPT_SYS(1);
```

17. SF_CHECK_CPT_SYS

定义：

```
int
SF_CHECK_CPT_SYS ()
```

功能说明：

系统的数据捕获环境启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统数据捕获环境的启用状态

```
SELECT SF_CHECK_CPT_SYS;
```

18. SF_SI

定义：

```
CHAR*
   SF_SI(
     index_sql  varchar(8188)
   )
```

功能说明：

输入索引创建语句，查看预计的执行信息

返回值：

索引创建的统计信息，排序区大小的建议值

举例说明：

```
SELECT SF_SI('create index idx_t1 on t1(c1,c2);');
```

19. SP_UNLOCK_USER

语法：

```
SP_UNLOCK_USER(
   USER_NAME varchar(128) not null
);
```

参数说明：

USER_NAME：需要解锁的用户名

功能说明：

为指定的用户解锁

举例说明：

```
CREATE USER USER123 IDENTIFIED BY USER123456;
--错误登录 3 次，导致用户被锁
SQL>LOGIN
服务名:
用户名:USER123456
密码:
端口号:
SSL 路径:
SSL 密码:
UKEY 名称:
UKEY PIN 码:
MPP 类型:
[-2501]:用户名或密码错误.

--解锁
--SYSDBA 用户登录:
call SP_UNLOCK_USER('USER123');
```

20. DUMP 函数

定义：

```
CHAR*
DUMP (
        Exp INT/BIGINT/DEC/DOUBLE/FLOAT/VARBINARY/
VARCHAR/DATE/TIME/DATETIME/TIME WITH TIME ZONE/TIMESTAMP WITH
TIME ZONE/INTERVAL YEAR/INTERVAL DAY/BLOB/CLOB ,
        Fmt  INT,
        Start  INT,
        Len  INT
)
```

功能说明：

获得表达式的内部存储字节

参数说明：

Exp：输入参数（必选），可以是任何基本数据类型

Fmt：输出格式进制，大于 1000 的情况下显示字符串的字符集、17 表示以字符显示

Start：开始进行返回的字节的位置

Len：需要返回的字节长度

返回值：

依次是数据类型、字节长度：内部存储字节（符号位、指数位、……）

对于大字段类型行内数据，字节长度为头长度+行内数据长度，内部存储字节只返

回大字段头+行内数据的全部字节。行外数据的字节长度为头长度，内部存储字节只返回大字段头。

举例说明：

例 1 查询字符串'an'的存储字节，显示字符集，显示成字符

```
select dump('an',1017);
```

查询结果：

```
Typ=2 Len=2 CharacterSet=GBK: a,n
```

例 2 查询日期类型'2005-01-01'的默认字节，默认进制显示

```
select dump(date '2005-01-01' ) ;
```

查询结果：

```
Typ=14 Len=12: 213,7,1,1,0,0,0,0,0,0,232,3
```

例 3 查询数字 123 的从第 2 个字节开始的 4 个字节，以 16 进制显示

```
select dump(1234567890,16,2,4);
```

查询结果：

```
Typ=7 Len=4: 2,96,49
```

例 4 查询 clob 类型的默认字节，默认进制显示。

1）行内数据：

```
create table testdump(c1 clob);
insert into testdump values('adbca');
insert into testdump values('1.曹雪芹，是中国文学史上最伟大也是最复杂的作家，《红楼梦》也是中国文学史上最伟大而又最复杂的作品。《红楼梦》写的是封建贵族的青年贾宝玉、林黛王、薛宝钗之间的恋爱和婚姻悲剧，而且以此为中心，写出了当时具有代表性的贾、王、史、薛四大家族的兴衰，其中又以贾府为中心，揭露了封建社会后期的种种黑暗和罪恶，及其不可克服的内在矛盾，对腐朽的封建统治阶级和行将崩溃的封建制度作了有力的批判，使读者预感到它必然要走向覆灭的命运。本书是一部具有高度思想性和高度艺术性的伟大作品，从本书反映的思想倾向来看，作者具有初步的民主主义思想，他对现实社会包括宫廷及官场的黑暗，封建贵族阶级及其家庭的腐朽，封建的科举制度、婚姻制度、奴婢制度、等级制度，以及与此相适应的社会统治思想即孔孟之道和程朱理学、社会道德观念等等，都进行了深刻的批判并且提出了朦胧的带有初步民主主义性质的理想和主张。这些理想和主张正是当时正在滋长的资本主义经济萌芽因素的曲折反映。2.曹雪芹，是中国文学史上最伟大也是最复杂的作家，《红楼梦》也是中国文学史上最伟大而又最复杂的作品。《红楼梦》写的是封建贵族的青年贾宝玉、林黛王、薛宝钗之间的恋爱和婚姻悲剧，而且以此为中心，写出了当时具有代表性的贾、王、史、薛四大家族的兴衰，其中又以贾府为中心，揭露了封建社会后期的种种黑暗和罪恶，及其不可克服的内在矛盾，对腐朽的封建统治阶级和行将崩溃的封建制度作了有力的批判，使读者预感到它必然要走向覆灭的命运。本书是一部具有高度思想性和高度艺术性的伟大作品，从本书反映的思想倾向来看，作者具有初步的民主主义思想，他对现实社会包括宫廷及官场的黑暗，封建贵族阶级及其家庭的腐朽，封建的科举制度、婚姻制度、奴婢制度、等级制度，以及与此相适应的社会统治思想即孔孟之道和程朱理学、社会道德观念等等，都进行了深刻的批判并且提出了朦胧的带有初步民主主义性质的理想和主张。这些理想和主张正是当时正在滋长的资本主义经济萌芽因素的曲折反映。');
select dump(c1) from testdump;
```

查询结果：

```
Typ=19 Len=48:
1,6,2,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,5,5,0,0,0,0,1,0,0,0,0,0,0,0,0,4,0,0,0,224,0,0,0,97,100,98,99,97
具体说明如下：
```

```
Len=48：大字段头 43 个字节 + 实际内容 5 个字节
1，大字段行内行外标记：行内
6, 0,0,0,0,0,0,0,大字段 id
5,0,0,0,大字段长度
0,0, 行外大字段的表空间 id
0,0, 行外大字段的文件 id
0,0,0,0，行外大字段的页 id
5,5,0,0，表 id
0,0,列 id
1,0,0,0,0,0,0,0，rowid
4,0，记录所在表空间号
0,0，记录所在文件号
224,0,0,0，记录所在页号
97,100,98,99,97 实际内容 5 个字节
```

2）行外数据：

可以使用 DM 数据库的各种编程接口插入 CLOB 列的行外数据，此处就不给出具体插入数据的例子，仅就查询结果各字节所表示的意义进行说明。

```
Typ=19 Len=43:
2,7,2,0,0,0,0,0,0,108,6,0,0,4,0,0,0,209,0,0,0,5,5,0,0,0,0,2,0,0,0,0,0,0,0
,4,0,0,0,224,0,0,0

具体说明如下：
Len=43：大字段头 43 个字节 + 实际内容 5 个字节
2,大字段行内行外标记：行外
7,2,0,0,0,0,0,0，大字段 id
108,6,0,0，大字段长度
4,0，行外大字段的表空间 id
0,0，行外大字段的文件 id
209,0,0,0，行外大字段的页 id
5,5,0,0，表 id
0,0，列 id
2,0,0,0,0,0,0,0，rowid
4,0，行外大字段对应表记录所在表空间号
0,0，行外大字段对应表记录所在文件号
224,0,0,0 行外大字段对应表记录所在页号
```

21. SP_CREATE_SYSTEM_PACKAGES

定义：

```
void
    SP_CREATE_SYSTEM_PACKAGES (
    CREATE_FLAG      int,
    PKGNAME          varchar(128)
)
```

功能说明：

单独创建或删除除了 DBMS_DBG、DMGEO 和 DBMS_JOB 以外的系统包

参数说明：

CREATE_FLAG：为 1 时表示创建除了 DBMS_DBG、DMGEO 和 DBMS_JOB 以外的指定系统包；为 0 时表示删除指定系统包

PKGNAME： 指定创建或删除的包名

返回值：

无

举例说明：

创建系统包 DBMS_SQL

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_SQL');
```

删除系统包 DBMS_SQL

```
SP_CREATE_SYSTEM_PACKAGES (0, 'DBMS_SQL');
```

22. SP_CLOSE_DBLINK

定义：

```
void
    SP_CLOSE_DBLINK(
    DBLINK        varchar(128)
)
```

功能说明：

关闭系统缓冲区中指定的空闲的外部链接，若指定的外部链接不处于空闲状态（可通过 V$DBLINK 查询），则不关闭。

参数说明：

DBLINK：指定的待关闭的外部链接名

返回值：

无

举例说明：

关闭之前创建的外部链接 LINK1。

```
SP_CLOSE_DBLINK('LINK1');
```

23. SP_XA_TRX_PROCESS

定义：

```
void
    SP_XA_TRX_PROCESS(
    TRXID         bigint,
    FLAG          int
)
```

功能说明：

指定提交或回滚 TRXID 对应的 XA TRX。指定的 TRXID 对应的 TRX 必须是 XA TRX，且处于游离状态即没有与会话相关联，否则报错。TRXID 可通过 V$TRX 查询。

参数说明：

TRXID：TRX 号

FLAG：0 表示回滚，1 表示提交

返回值：

无

24. GETUTCDATE

定义：

```
datetime
    GETUTCDATE ()
```

功能说明：

获取当前 UTC 时间

返回值：

当前 UTC 时间

举例说明：

获取当前 UTC 时间

```
SELECT GETUTCDATE();
```

25. SLEEP

定义：

```
VOID
    SLEEP(time dec)
```

功能说明：

表示让一个线程进入睡眠状态，等待一段时间 time 之后，该线程自动醒来进入到可运行状态。

参数说明：

time：睡眠时间，单位：秒

返回值：

无

举例说明：

让一个线程睡眠 1 秒钟之后，再醒过来继续运行

```
sleep(1);
```

15) 编目函数调用的系统函数

1. SF_GET_BUFFER_LEN

定义：

```
SF_GET_BUFFER_LEN(
    name varchar,
    length int,
    scale int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

length：列的精度，INT 类型

scale：列的刻度，INT 类型

功能说明：

根据某列数据类型名 name 和列的精度 length 刻度 scale 获取该列存贮在硬盘上的长度

返回值：

列存贮长度，数据类型为 INT

举例说明：

```
SELECT SF_GET_BUFFER_LEN('VARCHAR',3,2);
```

查询结果：

```
3
```

2. SF_GET_DATA_TYPE

定义：

```
SF_GET_DATA_TYPE(
    name varchar,
    scale int,
    version int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

scale：时间间隔类型刻度，INT 类型

version：版本号，INT 类型

功能说明：

根据数据类型关键字获取对应的 SQL 数据类型

返回值：

数据类型值，数据类型为 INT

举例说明：

```
SELECT SF_GET_DATA_TYPE('VARCHAR',3,2);
```

查询结果：

```
12
```

3. SF_GET_DATE_TIME_SUB

定义：

```
SF_GET_DATE_TIME_SUB(
    name varchar,
    scale int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

scale：类型刻度，INT 类型

功能说明：

获得时间类型的子类型

返回值：

时间类型子类型值，数据类型为 INT

举例说明：

```
SELECT SF_GET_DATE_TIME_SUB('datetime',2);
```

查询结果：

```
3
```

4. SF_GET_DECIMAL_DIGITS

定义：

```
SF_GET_DECIMAL_DIGITS(
    name varchar,
    scale int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

scale：预期类型刻度，INT 类型

功能说明：

根据某数据类型名和预期的刻度获取该数据类型的实际刻度

返回值：

类型实际刻度值，数据类型为 INT

举例说明：

```
SELECT SF_GET_DECIMAL_DIGITS('INT',2);
```

查询结果：

```
0
```

5. SF_GET_SQL_DATA_TYPE

定义：

```
SF_GET_SQL_DATA_TYPE(
    name varchar
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

功能说明：

根据某数据类型名返回该数据类型的 SQL 数据类型值

返回值：

SQL 数据类型值，数据类型为 INT

举例说明：

```
SELECT SF_GET_SQL_DATA_TYPE('INT');
```

查询结果：

```
4
```

6. SF_GET_SYS_PRIV

定义：

```
SF_GET_SYS_PRIV(
    privid int
)
```

参数说明：

privid：数据类型为 INT。取值范围：数据库权限 4096-4244，对象权限 8192-8198。除了 4123、4204 和 4205

功能说明：

获得 privid 所代表的权限操作

返回值：

前导字符串，数据类型为 VARCHAR

举例说明：

```
SELECT SF_GET_SYS_PRIV(4096);
SELECT SF_GET_SYS_PRIV(4099);
```

查询结果：

```
CREATE DATABASE
CREATE LOGIN
```

7. SF_GET_OCT_LENGTH

定义：

```
SF_GET_OCT_LENGTH(
    name varchar,
    length int
)
```

参数说明：

name：数据类型名，数据类型为 VARCHAR

length：类型长度，INT 类型

功能说明：

返回变长数据类型的长度

返回值：

类型长度，数据类型为 INT

举例说明：

```
SELECT SF_GET_OCT_LENGTH('VARCHAR',3);
```

查询结果：

```
3
```

8. SF_GET_TABLES_TYPE

定义：

```
SF_GET_TABLES_TYPE(
    type varchar
)
```

参数说明：

type：表的类型名，数据类型为 VARCHAR

功能说明：

返回表类型名

返回值：

表类型名，数据类型为 VARCHAR

举例说明：

```
SELECT SF_GET_TABLES_TYPE('UTAB');
```

查询结果：

```
TABLE
```

9. CURRENT_SCHID

定义：

CURRENT_SCHID()

参数说明：

无

功能说明：

返回当前会话的当前模式 ID

返回值：

当前会话的当前模式 ID

举例说明：

```
SELECT CURRENT_SCHID();
```

查询结果：

```
150994945
```

10. SF_GET_SCHEMA_NAME_BY_ID

定义：

```
SF_GET_SCHEMA_NAME_BY_ID(
    schid int
)
```

参数说明：

schid：模式 ID，INT 类型

功能说明：

根据模式 ID 返回模式名

返回值：

模式名，数据类型为 VARCHAR

举例说明：

```
SELECT SF_GET_SCHEMA_NAME_BY_ID(150994945);
```

查询结果：

```
SYSDBA
```

11. SF_COL_IS_IDX_KEY

定义：

```
SF_COL_IS_IDX_KEY(
    key_num int,
    key_info varbinary,
    col_id int
)
```

参数说明：

key_num：键值个数，数据类型为 INT

key_info：键值信息，数据类型为 VARBINARY

col_id：列 ID，INT 类型

功能说明：

判断所给的 colid 是否是 index key

返回值：

是否索引键，数据类型为 INT

举例说明：

```
CREATE TABLE TT1(C1 INT);
CREATE INDEX ID1 ON TT1(C1);
SELECT ID FROM SYSOBJECTS WHERE NAME LIKE 'TT1';//1334
SELECT INDS.KEYNUM, INDS.KEYINFO, SF_COL_IS_IDX_KEY(INDS.KEYNUM, INDS.KEYINFO, COLS.COLID) FROM (SELECT ID,PID,NAME FROM SYSOBJECTS WHERE SUBTYPE$='INDEX') AS OBJ_INDS, SYSCOLUMNS AS COLS, SYSINDEXES AS INDS WHERE COLS.ID = 1334 AND OBJ_INDS.PID = 1334 AND INDS.ID = OBJ_INDS.ID;
```

查询结果：

```
1  000041  1     --ID1
0          0     --聚集索引
```

12. SF_GET_INDEX_KEY_ORDER

定义：

```
SF_GET_INDEX_KEY_ORDER(
    key_num int,
    key_info varbinary,
    col_id int
)
```

参数说明：

key_num：索引键个数，数据类型为 INT

key_info：键值信息，数据类型为 VARBINARY

col_id：列 ID，INT 类型

功能说明：

获得当前 column 的 key 的排序

返回值：

类型长度，数据类型为 VARCHAR

举例说明：

```
CREATE TABLE TT1(C1 INT);
CREATE INDEX ID1 ON TT1(C1);
SELECT ID FROM SYSOBJECTS WHERE NAME LIKE 'TT1';//1135
SELECT SF_GET_INDEX_KEY_ORDER(INDS.KEYNUM, INDS.KEYINFO, COLS.COLID) FROM (SELECT ID,PID,NAME FROM SYSOBJECTS WHERE SUBTYPE$='INDEX') AS OBJ_INDS, SYSCOLUMNS AS COLS, SYSINDEXES AS INDS WHERE COLS.ID = 1135 AND OBJ_INDS.PID = 1135 AND INDS.ID = OBJ_INDS.ID;
```

查询结果：

```
A
NULL
```

13. SF_GET_INDEX_KEY_SEQ

定义：

```
SF_GET_INDEX_KEY_SEQ(
    key_num int,
    key_info varbinary,
    col_id int
```

)

参数说明：

key_num：索引键个数，数据类型为 INT

key_info：键值信息，数据类型为 VARBINARY

col_id：列 ID，INT 类型

功能说明：

获得当前 column 所在的 key 序号

返回值：

当前列 KEY 序号，数据类型为 INT

举例说明：

```
CREATE TABLE TT1(C1 INT);
CREATE INDEX ID1 ON TT1(C1);
SELECT ID FROM SYSOBJECTS WHERE NAME LIKE 'TT1';//1135
SELECT SF_GET_INDEX_KEY_SEQ(INDS.KEYNUM, INDS.KEYINFO, COLS.COLID) FROM
(SELECT ID,PID,NAME FROM SYSOBJECTS WHERE SUBTYPE$='INDEX') AS OBJ_INDS,
SYSCOLUMNS AS COLS, SYSINDEXES AS INDS WHERE COLS.ID = 1135 AND OBJ_INDS.PID =
1135 AND INDS.ID = OBJ_INDS.ID;
```

查询结果：

```
1
-1
```

14. SF_GET_UPD_RULE

定义：

```
SF_GET_UPD_RULE(
    rule varchar(2)
)
```

参数说明：

rule：规则名，数据类型为 VARCHAR，rule 参数只会用到 rule[0]，取值为’’/’C’/’N’/’D’，分别表示 no act/cascade/set null/set default。

功能说明：

解析在 SYSCONS 中 faction 中保存的外键更新规则

返回值：

外键更新规则值，数据类型为 INT

举例说明：

```
SELECT SF_GET_UPD_RULE('C');
```

查询结果：

```
0
```

15. SF_GET_DEL_RULE

定义：

```
SF_GET_DEL_RULE(
    rule varchar(2)
)
```

参数说明：

rule：规则名，数据类型为 VARCHAR，rule 参数只会用到 rule[1]，取值为’’/’C’/’N’/’D’，分别表示 no act/cascade/set null/set default。

功能说明：

解析在 SYSCONS 中 faction 中保存的外键删除规则

返回值：

外键删除规则值，数据类型为 INT

举例说明：

```
SELECT SF_GET_DEL_RULE('AC');
```

查询结果：

```
0
```

16. SF_GET_OLEDB_TYPE

定义：

```
SF_GET_OLEDB_TYPE(
    name varchar
)
```

参数说明：

name：类型名，数据类型为 VARCHAR

功能说明：

获得 OLEDB 的数据类型长度

返回值：

数据类型值，数据类型为 INT

举例说明：

```
SELECT SF_GET_OLEDB_TYPE('INT');
```

查询结果：

```
3
```

17. SF_GET_OLEDB_TYPE_PREC

定义：

```
SF_GET_OLEDB_TYPE_PREC(
    name varchar,
    length int   )
```

参数说明：

name：类型名，数据类型为 VARCHAR

length：类型长度，数据类型为 INT

功能说明：

获得 OLEDB 的数据类型的精度

返回值：

数据类型的精度值，数据类型为 INT

举例说明：

```
SELECT SF_GET_OLEDB_TYPE_PREC('INT',2);
```

查询结果：

```
10
```

18. SP_GET_TABLE_COUNT

定义：

```
SP_GET_TABLE_COUNT(
    table_id int  )
```

参数说明：

table_id：类型长度，数据类型为 INT

功能说明：

获得表行数

返回值：

表的行数，数据类型为 INT

举例说明：

```
SELECT SP_GET_TABLE_COUNT(1097);
```

查询结果：

```
69
```

19. SF_OLEDB_TYPE_IS_LONG

定义：

```
SF_OLEDB_TYPE_IS_LONG(
    typename varchar   )
```

参数说明：

typename：类型名，数据类型为 VARCHAR

功能说明：

判断类型是否较长

返回值：

0 或者 1，数据类型为 INT

举例说明：

```
SELECT SF_OLEDB_TYPE_IS_LONG('IMAGE');
SELECT SF_OLEDB_TYPE_IS_LONG('INT');
```

查询结果：

```
1
0
```

20. SF_OLEDB_TYPE_IS_BESTMATCH

定义：

```
SF_OLEDB_TYPE_IS_BESTMATCH(
    typename varchar
)
```

参数说明：

typename：类型名，数据类型为 VARCHAR

功能说明：

判断类型是否精确匹配类型

返回值：

0 或者 1，数据类型为 INT

举例说明：

```
SELECT SF_OLEDB_TYPE_IS_BESTMATCH('INT');
SELECT SF_OLEDB_TYPE_IS_BESTMATCH('IMAGE');
```

查询结果：

```
1
0
```

21. SF_OLEDB_TYPE_IS_FIXEDLEN

定义：

```
SF_OLEDB_TYPE_IS_FIXEDLEN(
    typename varchar  )
```

参数说明：

typename：类型名，数据类型为 VARCHAR

功能说明：

判断类型是否为定长类型

返回值：

0 或者 1，数据类型为 INT

举例说明：

```
SELECT SF_OLEDB_TYPE_IS_FIXEDLEN('INT');
SELECT SF_OLEDB_TYPE_IS_FIXEDLEN('IMAGE');
```

查询结果：

```
1
0
```

22. SF_GET_TABLE_COUNT

定义：

```
SP_GET_TABLE_COUNT(
    table_id int  )
```

参数说明：

table_id：类型长度，数据类型为 INT

功能说明：

获得表行数。功能和 SP_GET_TABLE_COUNT 一样

返回值：

表的行数，数据类型为 INT

举例说明：

```
SELECT SP_GET_TABLE_COUNT(1097);
```

查询结果：

```
69
```

23. SF_GET_TABLE_COUNT

定义：

```
SP_GET_TABLE_COUNT(
    schema_name varchar(128),
    table_name varchar(128))
```

参数说明：

schema_name：模式名，数据类型为 varchar

功能说明：

获得表行数。功能和 SP_GET_TABLE_COUNT 一样

返回值：

表的行数，数据类型为 INT

举例说明：

```
SELECT SP_GET_TABLE_COUNT('SYSDBA','T');
```

查询结果：

```
69
```

16) BFILE

1. BFILENAME

定义：

```
BFILE
BFILENAME(
    dir      VARCHAR,
    filename   VARCHAR
)
```

参数说明：

dir：数据库的目录对象名，字符串中不能包含":"

filename：操作系统文件名，字符串中不能包含":"

功能说明：

生成一个 BFILE 类型数据

返回值：

BFILE 数据类型对象

17) 定制会话级 INI 参数

提供用户定制会话级 ini 参数默认值的功能。定制以后，当用户再次登录时无需复杂的设置就可以在当前会话中使用定制的会话级 ini 参数，并且使无法设置参数的 B/S 或者 C/S 应用也能使用特定的会话级 ini 参数。

定制后的 ini 参数值可以通过数据字典 SYSUSERINI 查看。

使用时有以下限制：

1) 用户对会话级参数的定制和取消对于当前会话不会立刻生效，也不会影响当前已经连接的该用户的其他会话。该用户新登录的会话会使用定制的值作为默认值。
2) MPP 环境下，参数的定制在整个 MPP 环境所有的服务器节点生效。
3) 对于一个会话级 ini 参数，其取值优先级顺序为： 会话中设置的值 > 用户定制的值 > INI 文件中匹配的值。
4) DBA 用户拥有定制、查询和删除所有用户 ini 的权限；普通用户拥有定制、查询和删除自身 ini 的权限。

对于不同数据类型的参数要使用不同的系统过程。ini 参数的 VALUE 有三种类型：INT、

VARCHAR 和 DOUBLE，分别对应三个系统过程：

1. SP_SET_USER_INI

定义：

```
SP_SET_USER_INI(
USER VARCHAR,
PARA_NAME VARCHAR,
VALUE INT
);
```

功能说明：

定制 INT 类型的 ini 参数

参数说明：

USER：用户名

PARA_NAME：参数名

VALUE：INT 类型参数值

举例说明：

```
SP_SET_USER_INI('USER1','SORT_BUF_SIZE','1234');
```

2. SP_SET_USER_STRING_INI

定义：

```
SP_SET_USER_STRING_INI(
USER VARCHAR,
PARA_NAME VARCHAR,
VALUE VARCHAR
);
```

功能说明：

定制 VARCHAR 类型的 ini 参数。

参数说明：

USER：用户名

PARA_NAME：参数名

VALUE:VARCHAR 类型参数值

3. SP_SET_USER_DOUBLE_INI

定义

```
SP_SET_USER_STRING_INI(
USER VARCHAR,
PARA_NAME VARCHAR,
VALUE VARCHAR
);
```

功能说明：

定制 DOUBLE 类型的 ini 参数。

参数说明：

USER：用户名

PARA_NAME：参数名

VALUE:DOUBLE 类型参数值

举例：

```
SP_SET_USER_DOUBLE_INI('USER2','INDEX_SKIP_SCAN_RATE','0.0035');
```

4. SP_CLEAR_USER_INI

定义

```
SP_CLEAR_USER_INI(
USER VARCHAR,
PARA_NAME VARCHAR
);
```

功能说明：

清除用户 USER 的 PARA_NAME 的参数定制。当 USER 和 PARA_NAME 都等于 NULL 时，表示清除所有用户的所有定制；当 PARA_NAME 为 NULL 时，表示清除用户 USER 的所有定制。不支持 USER 为 NULL，PARA_NAME 不为 NULL 的情况。

参数说明：

USER：用户名

PARA_NAME：参数名

举例说明：

```
SP_CLEAR_USER_INI('USER2',NULL);
```

18) 为 SQL 指定 HINT

提供无需修改 SQL 语句但依然能按照指定的 HINT 运行语句的相关功能。

使用时有以下限制：

1） INI 参数 ENABLE_INJECT_HINT 需设置为 1；
2） SQL 只能是语法正确的增删改查语句；
3） SQL 会经过系统格式化，格式化之后的 SQL 和指定的规则名称必须全局唯一；
4） HINT 一指定，则全局生效；
5） 系统检查 SQL 匹配时，必须是整条语句完全匹配，不能是语句中子查询匹配；
6） 可通过 SYSINJECTHINT 视图查看已指定的 SQL 语句和对应的 HINT。

1. SF_INJECT_HINT

定义

```
VARCHAR
    SF_INJECT_HINT (
        SQL_TEXT        TEXT,
        HINT_TEXT       TEXT,
        NAME            VARCHAR(128),
        DESCRIPTION     VARCHAR(256),
        VALIDATE        BOOLEAN
)
```

功能说明：

对指定 SQL 增加 HINT。

参数说明：

SQL_TEXT： 要指定 HINT 的 SQL 语句

HINT_TEXT：要为 SQL 指定的 HINT
NAME：可以指定名称，或者设为 NULL 让系统自动创建名称
DESCRIPTION：对规则的详细描述，可为 NULL
VALIDATA：规则是否生效，可为 NULL，则为默认值 TRUE

返回值：

执行成功返回名称，执行失败报错误信息

举例说明：

为以下语句指定 HINT 为 MMT_SIZE = 4：

```
SF_INJECT_HINT('SELECT * FROM A;', 'MMT_SIZE(4)', 'TEST_INJECT', 'to test function of injecting hint', TRUE);
```

2. SF_DEINJECT_HINT

定义：

```
INT
SF_DEINJECT_HINT (
    NAME          VARCHAR(128)
)
```

功能说明：

对指定 SQL 撤回已增加的 HINT

参数说明：

NAME： 要删除的规则名称

返回值：

执行成功返回 0，执行失败返回错误码

举例说明：

为 SQL 撤回已指定的 HINT：

```
SF_DEINJECT_HINT('TEST_INJECT');
```

3. SF_ALTER_HINT

定义：

```
INT
SF_ALTER_HINT (
    NAME             VARCHAR(128),
    ATTRIBUTE_NAME   VARCHAR(12),
    ATTRIBUTE_VALUE  VARCHAR(256)
)
```

功能说明：

修改已指定 HINT 的规则属性

参数说明：

NAME： 要修改的规则名称
ATTRIBUTE_NAME：要修改的属性名
STATUS:规则是否生效，取值 ENABLED/DISABLED
NAME:规则名称
DESCRIPTION:规则的详细描述
ATTRIBUTE_VALUE：设置的属性值

返回值：

执行成功返回 0，执行失败返回错误码

举例说明：

让已指定的 HINT 的规则失效：

```
SF_ALTER_HINT('TEST_INJECT', 'STATUS', 'DISABLED');
```

19) 时区设置

本小节的过程与函数都是用来设置时区相关信息。

1. SP_SET_TIME_ZONE

定义：

VOID SP_SET_TIME_ZONE(interval day)

VOID SP_SET_TIME_ZONE(interval day to hour)

VOID SP_SET_TIME_ZONE(interval day to minute)

VOID SP_SET_TIME_ZONE(interval day to second)

VOID SP_SET_TIME_ZONE(interval hour)

VOID SP_SET_TIME_ZONE(interval hour to minute)

SP_SET_TIME_ZONE(interval hour to second)

功能说明：

设置时区为标准时区加上 interval day 中的小时和分钟。

返回值：

无。

举例说明：

```
SP_SET_TIME_ZONE(INTERVAL '12 3:9' hour to second);
drop table ttime;
create table ttime (c2 timestamp with time zone);
insert into ttime values('1977-01-10 8:0:0');
select * from ttime;
```

查询结果：

```
1977-01-10 08:00:00.000000 +12:03
```

2. SP_SET_TIME_ZONE_STRING

定义：

VOID SP_SET_TIME_ZONE_STRING(VARCHAR)

功能说明：

设置时区为标准时区加上 varchar 的 interval hour。varchar 只能是 interval hour 或 hour to minute。

返回值：

无。

举例说明：

```
SP_SET_TIME_ZONE_string('7  3');
drop table ttime;
create table ttime (c2 timestamp with local time zone);
insert into ttime values('1977-01-10 8:0:0 +4:00');
```

```
select * from ttime;
```

查询结果：

```
1977-01-10 11:03:00.000000
```

3. SP_SET_TIME_ZONE_LOCAL

定义：

VOID SP_SET_TIME_ZONE_LOCAL()

功能说明：

设置时区为当前服务器的时区。

返回值：

无。

举例说明：

```
SP_SET_TIME_ZONE_LOCAL();
drop table ttime;
create table ttime (c2 timestamp with local time zone);
insert into ttime values('1977-01-10 8:0:0 +3:00');
select * from ttime;
```

查询结果：

```
1977-01-10 13:00:00.000000
```

4. SF_TIME_ADD_TIME_ZONE_INTERVAL/
SF_DATETIME_ADD_TIME_ZONE_INTERVAL

定义：

VOID SF_TIME_ADD_TIME_ZONE_INTERVAL(time)

或

VOID SF_TIME_ADD_TIME_ZONE_INTERVAL(datetime)

功能说明：

将指定时间加上 LOCAL 时区时间。

返回值：

无。

举例说明：

```
select SF_TIME_ADD_TIME_ZONE_INTERVAL('1988-01-01 2:00'); 或
select SF_DATETIME_ADD_TIME_ZONE_INTERVAL('1988-01-01 2:00:36');
```

查询结果：

```
1988-01-01 10:00:00  或
1988-01-01 10:00:36
```

20) XML

本小节的过程与函数都是用来解析和查询 XML 数据的，目前 DM 支持解析的 XML 数据大小不得超过 500M。

1. SF_XMLQUERY

定义：

clob

```
SF_XMLQUERY(
    xmldata   clob,
    xpath     clob
)
```

功能说明：

解析 xmldata 并查询指定路径 xpath 下的数据。

参数说明：

xmldata：待解析的 XML 数据

xpath：指定的解析路径

返回值：

解析得到的 XML 数据。

举例说明：

```
SELECT SF_XMLQUERY('<a><b>b</b></a>', '/a');
```

查询结果：

```
<a><b>b</b></a>
```

2. XMLQUERY

定义：

```
clob
XMLQUERY(
    xmldata   clob,
    xpath     varchar2(8188)
)
```

功能说明：

解析 xmldata 并查询指定路径 xpath 下的数据。

参数说明：

xmldata：待解析的 XML 数据

xpath：指定的解析路径

返回值：

解析得到的 XML 数据。

举例说明：

```
SELECT XMLQUERY('<a><b>b</b></a>', '/a');
```

查询结果：

```
<a><b>b</b></a>
```

3. EXISTSNODE

定义：

```
int
EXISTSNODE(
    xmldata   clob,
    xpath     varchar2(8188)
)
```

功能说明：

xmldata 中是否存在指定的路径节点，若存在则返回 1，不存在返回 0。xmldata

或 xpath 为 NULL 或者空串的情况下一律返回 0。

参数说明：

xmldata：待解析的 XML 数据

xpath：指定的路径节点

返回值：

1 代表节点存在，0 代表节点不存在。

举例说明：

```
SELECT EXISTSNODE('<a><b>b</b></a>', '/a');
```

查询结果：

```
1
```

4. EXTRACTVALUE

定义：

```
varchar2
EXTRACTVALUE(
    xmldata   clob,
    xpath     varchar2(8188)
)
```

功能说明：

获取 xmldata 中指定路径下的结点值，若路径下不止一个结点，或者结点不是叶子结点，将报错处理。

参数说明：

xmldata：待解析的 XML 数据

xpath：指定的路径

返回值：

获取的节点值。

举例说明：

```
SELECT EXTRACTVALUE('<a><b>b</b></a>', '/a/b');
```

查询结果：

```
b
```

5. APPENDCHILDXML

定义：

```
clob
APPENDCHILDXML(
    xmldata   clob,
    xpath     varchar2(8188),
    child     clob
)
```

功能说明：

将 child 结点插入到 xmldata 的 xpath 下的结点中，并将插入结点后的新 xmldata 返回。若 xmldata 为 NULL 或空串，函数将报错；若 xpath 为 NULL 或空串，函数将返回 NULL；若 child 为 NULL 或者空串，函数将返回 xmldata 原串内容。

参数说明：

xmldata：源 XML 数据

xpath：指定的插入节点路径

child：指定的待插入节点

返回值：

插入节点后的新 XML 数据。

举例说明：

```
SELECT APPENDCHILDXML('<a><b>b</b></a>', '/a', '<d>xxx<d>');
```

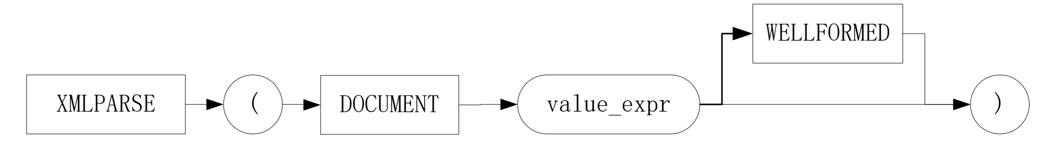
查询结果：

```
<a><b>b</b><d>xxx</d></a>
```

6. XMLPARSE

定义：

XMLPARSE 的语法如下：



功能说明：

XMLPARSE 用于解析 XMLTYPE 类型数据，亦即 value_expr 的值。若指定了 WELLFORMED 参数，则不对 XML 内容进行检查，否则会对内容的合法性进行检查，内容不合法则报错。

举例说明：

```
SELECT XMLPARSE(DOCUMENT '<a>good</a>' WELLFORMED);
```

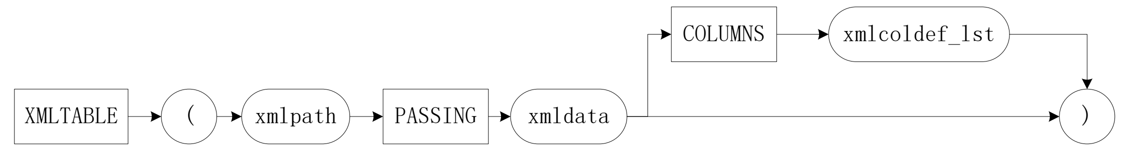
查询结果：

```
<a>good</a>
```

7. XMLTABLE

定义：

XMLTABLE 的语法如下：



功能说明：

查询 XML 数据的子选项。

参数说明：

xmlpath：XML 数据的路径

xmldata：XMLTYPE 类型数据

xmlcoldef_lst：列定义列表

举例说明：

```
CREATE TABLE T3(C1 VARCHAR2);
INSERT INTO T3 VALUES('<A><标签 属性="属性名">HELLO</标签></A>');
SELECT X1,X2 FROM T3, XMLTABLE('/A/标签' PASSING T3.C1 COLUMNS X1 VARCHAR2(12) PATH '@属性', X2 VARCHAR2(12) PATH 'text()');
```

查询结果：

```
行号       X1     X2
------- ------ -----
1          属性名 HELLO
```

附录 4 DM 技术支持

如果您在安装或使用 DM 及其相应产品时出现了问题，请首先访问我们的 Web 站点 http://www.dameng.com/。在此站点我们收集整理了安装使用过程中一些常见问题的解决办法，相信会对您有所帮助。

您也可以通过以下途径与我们联系，我们的技术支持工程师会为您提供服务。

达梦数据库（武汉）有限公司

地址：武汉市东湖高新技术开发区高新大道 999 号未来科技大厦 C3 栋 16-19 层

邮编：430073

电话：(+86)027-87588000

传真：(+86)027-87588000-8039

达梦数据库（北京）有限公司

地址：北京市海淀区北三环西路 48 号数码大厦 B 座 905

邮编：100086

电话：(+86)010-51727900

传真：(+86)010-51727983

达梦数据库（上海）有限公司

地址：上海市闸北区江场三路 28 号 301 室

邮编：200436

电话：(+86)021-33932716

传真：(+86)021-33932718

地址：上海市浦东张江高科技园区博霞路 50 号 403 室

邮编：201203

电话：（+86）021-33932717

传真：（+86）021-33932717-801

达梦数据库（广州）有限公司

地址：广州市荔湾区中山七路 330 号荔湾留学生科技园 703 房

邮编：510145

电话：(+86)020-38371832

传真：(+86)020-38371832

达梦数据库（海南）有限公司

地址：海南省海口市玉沙路富豪花园 B 座 1602 室

邮编：570125

电话：(+86)0898-68533029

传真：(+86)0898-68531910

达梦数据库（南宁）办事处

地址：广西省南宁市科园东五路四号南宁软件园五楼

邮编：530003
电话：（+86）0771-2184078
传真：（+86）0771-2184080

达梦数据库（合肥）办事处

地址：合肥市包河区马鞍山路金帝国际城 7 栋 3 单元 706 室
邮编：230022
电话：（+86）0551-3711086

达梦数据库（深圳）办事处

地址：深圳市福田区皇岗路高科利大厦 A 栋 24E
邮编：518033
电话：0755-83658909
传真：0755-83658909

技术服务：

电话：400-991-6599
邮箱：dmtech@dameng.com



武汉达梦数据库有限公司

总部：武汉市东湖高新技术开发区高新大道999号
未来科技大厦C3栋16—19层

电话：(+86) 027-87588000

